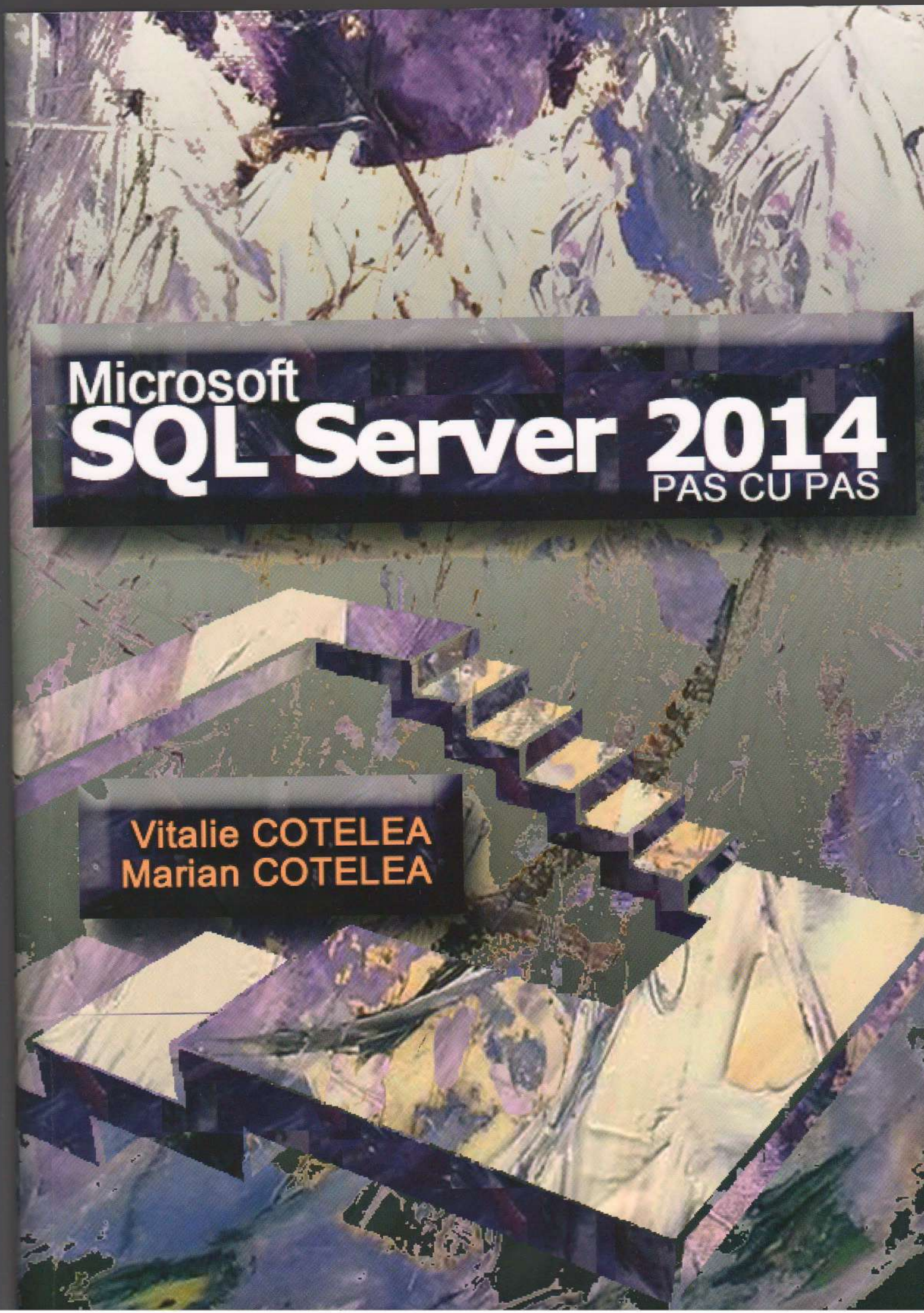
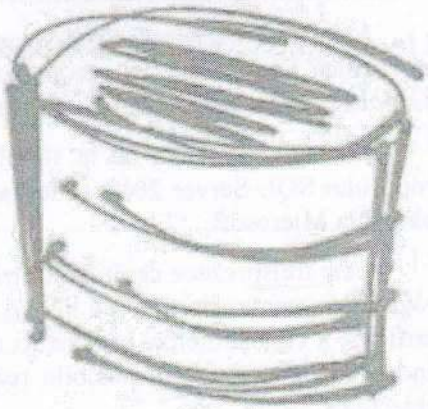

Microsoft
SQL Server 2014
PAS CU PAS
Vitalie COTELEA
Marian COTELEA

ACADEMIA DE STUDII ECONOMICE A MOLDOVEI
Catedra „Cibernetică şi Informatică Economică”



Vitalie Cotelea
Marian Cotelea

Microsoft SQL Server 2014

PAS CU PAS

Chişinău-2015

CZU 004.65(075.8)
C 78

Manualul ***Microsoft SQL Server 2014. Pas cu pas*** a fost elaborat în cadrul catedrei „Cibernetică şi Informatică Economică" şi recomandat pentru editare de ***Senatul Academiei de Studii Economice din Moldova*** (proces verbal nr.6 din 29 aprilie 2015).

Lucrarea prezintă, de la simplu la complex, cunoştinţele teoretice şi practice necesare utilizării produsului SQL Server 2014 - ultima versiune a sistemului pentru gestiunea bazelor de date, elaborat de compania Microsoft.

Cele treisprezece capitole ale volumului de faţă nu necesită niciun fel de cunoştinţe preliminare de programare, sunt scrise într-un limbaj simplu, accesibil, neacademic, didactic, oferă o modalitate rapidă de verificare a cunoştinţelor, prin seturi de întrebări şi exerciţii de la sfârşitul capitolelor, pentru majoritatea fiind propuse mai multe posibile rezolvări (cu exemple practice şi concrete) pe parcursul dezvoltării subiectelor.

Chiar dacă tematica cuprinsă între copertele acestei lucrări nu are un caracter exhaustiv, ceea ce s-ar dori prin intermediul unui tutorial, el prezintă principalele concepte şi mecanisme cu ajutorul cărora se pot efectua proiectarea, mentenanţa şi restructurarea bazelor de date. Alegerea variantelor de soluţii şi aplicarea eficientă a instrumentelor descrise rămâne la latitudinea şi depinde de imaginaţia şi nevoile personale ale fiecărui utilizator care dezvoltă un proiect.

Volumul de faţă se adresează: studenţilor, oricărei persoane care doreşte să se iniţieze în domeniul bazelor de date, angajaţilor companiilor de software sau potenţialilor angajaţi, care nu sunt neapărat programatori, dar lucrează, într-un fel sau altul, cu baze de date, implementatorilor, consultanţilor, instructorilor, analiştilor, administratorilor de sistem sau de baze de date, programatorilor care nu au luat contact cu bazele de date şi doresc o familiarizare rapidă cu SQL Server 2014.

Recenzenţi:

Prof. univ., dr. hab. în informatică Ilie Costaş
Conf. cercet., dr. hab. în informatică Svetlana Cojocaru

Descrierea CIP a Camerei Naţionale a Cărţii

Cotelea, Vitalie
Microsoft SQL Server 2014: Pas cu pas / Acad. de Studii Econ. a Moldovei. Catedra „Cibernetică şi Informatică Economică" – Chişinău: Întreprinderea de Stat. Firma editorial-poligrafică „Tipografia Centrală", 2015. – 278 p.
Bibliogr.: p. 278 (10 tit.). – 100 ex.
ISBN 978-9975-75-735-5.

CZU 004.65(075.8)
C 78

Autori:

© **Vitalie Cotelea (ASEM)**
© **Marian Cotelea (Endava)**

ISBN 978-9975-75-735-5.

Întreprinderea de Stat. Firma editorial-poligrafică „Tipografia Centrală"

CUPRINS

INTRODUCERE **9**

1. INSTALAREA ŞI CONFIGURAREA SQL SERVER 2014 ..... **11**

1.1. SISTEMUL SQL SERVER 2014 11
1.1.1. Baze de date client-server *11*
1.1.2. Facilităţi noi oferite de SQL Server 2014 ..... *11*
1.1.3. Componentele logice ale sistemului SQL Server *12*
1.2. INSTALAREA ŞI CONFIGURAREA SISTEMULUI 14
1.2.1. Capacităţile hardware şi software necesare *14*
1.2.2. Instalarea sistemului *14*
1.2.3. Configurarea sistemului *21*
1.2.4. Iniţializarea serviciilor *27*
1.3. CONEXIUNEA LA SERVERE 29
1.3.1. Înregistrarea serverelor *29*
1.3.2. Excluderea unui server *33*
1.3.3. Modificarea înregistrării unui server *33*
1.4. LOGINURI ÎN SISTEMUL SQL SERVER 33
1.4.1. Autentificarea Windows *33*
1.4.2. Autentificarea mixtă *34*
1.4.3. Configurarea modului de autentificare *34*
1.4.4. Crearea loginului propriu *34*
1.5. GESTIONAREA ROLURILOR 36

2. CREAREA ŞI ÎNTREŢINEREA BAZEI DE DATE **41**

2.1. FIŞIERELE GESTIONATE DE SQL SERVER 2014 41
2.2. CREAREA BAZEI DE DATE 42
2.2.1. Lansarea SQL Server Management Studio pentru crearea bazei de date *42*
2.2.2. Afişarea şi modificarea proprietăţilor bazei de date (General) *44*
2.2.3. Consultarea şi modificarea proprietăţilor bazei de date (Options) *46*
2.2.4. Afişarea şi adăugarea grupurilor de fişiere *51*
2.3. BAZE DE DATE DE SISTEM 54
2.4. CREAREA PLANULUI DE ÎNTREŢINERE 54
2.4.1. Crearea planului de întreţinere cu ajutorul asistentului *54*
2.4.2. Crearea manuală a planului de întreţinere *57*

3. CREAREA ŞI MODIFICAREA TABELELOR ÎN SQL SERVER MANAGEMENT STUDIO **61**

3.1. TIPURI DE DATE 61
3.1.1. Date numerice exacte *61*
3.1.2. Date numerice aproximative *62*
3.1.3. Date secvenţe de caractere *62*
3.1.4. Date binare *62*
3.1.5. Tipuri de date dată/oră *63*
3.1.6. Tipuri de date speciale *63*
3.1.7. Date UNICODE *64*

3.1.8. Ordinea de precedență 64
3.2. CONSTRÂNGERI DE INTEGRITATE 65
3.3. CREAREA TABELELOR 66
3.3.1. Un exemplu cu patru tabele 66
3.3.2. Definirea proprietăților tabelului 68
3.3.3. Definirea coloanelor 68
3.3.4. Proprietățile coloanelor 68
3.3.5. Stabilirea cheii primare 71
3.3.6. Salvarea tabelului 72
3.4. ȘTERGEREA, REDENUMIREA ȘI ACTUALIZAREA TABELELOR 72
3.4.1. Ștergerea tabelelor din baza de date 72
3.4.2. Redenumirea tabelelor 72
3.4.3. Introducerea, editarea și ștergerea datelor din tabele 72
3.5. MODIFICAREA STRUCTURII UNUI TABEL 73
3.5.1. Modificarea proprietăților coloanelor 73
3.5.2. Inserarea unui câmp nou 74
3.5.3. Ștergerea unui câmp existent 74
4. INSTRUCȚIUNEA SELECT TRANSACT-SQL 77
4.1. ÎNCĂRCAREA TRANSACT-SQL QUERY EDITOR 77
4.2. FUNCȚIONALITĂȚILE COMPONENTEI TRANSACT-SQL QUERY EDITOR 78
4.3. ELEMENTE DE BAZĂ ALE INSTRUCȚIUNII SELECT 79
4.3.1. Operatori și expresii utilizate în interogări 80
4.3.2. Interogări simple 81
4.3.3. Funcții pe un sigur rând 83
4.3.4. Funcții de grup 85
4.3.5. Limitarea numărului de linii returnate 87
4.4. INTEROGĂRI CU JONCȚIUNI 88
4.4.1. Interogări cu joncțiuni interne 89
4.4.2. Interogări cu joncțiuni externe 89
4.4.3. Produsul Cartezian 91
4.5. SUBINTEROGĂRI ȘI OPERATORI PE MULȚIMI 92
4.5.1. Subinterogări corelate și necorelate 92
4.5.2. Subinterogări ce întorc un scalar sau o linie 93
4.5.3. Subinterogări cu operatorul IN 94
4.5.4. Subinterogări cu operatorul ANY 95
4.5.5. Subinterogări cu operatorul ALL 96
4.5.6. Subinterogări cu operatorul EXISTS 96
4.5.7. Subinterogări în clauza FROM 97
4.5.8. Interogări cu operatori din teoria mulțimilor 98
4.6. STRUCTURI DE CONTROL ALE FLUXULUI 99
4.6.1. Blocuri Transact-SQL 99
4.6.2. Inițializarea variabilelor locale 100
4.6.3. Structura selectivă CASE 101
4.6.4. Structura alternativă IF...ELSE 102
4.6.5. Structura repetitivă WHILE 103
4.6.6. Instrucțiunea BREAK 104

4.6.7. Instrucțiunea CONTINUE ... *104*
4.6.8. Instrucțiunea RETURN ... *105*
4.6.9. Goto-uri și etichete ... *106*
4.6.10. Operatorului WAITFOR ... *106*

5. CREAREA TABELELOR ȘI INDECȘILOR ... **111**

5.1. CREAREA ȘI MODIFICAREA STRUCTURII TABELELOR ÎN TRANSACT-SQL ... 111
5.1.1. Instrucțiunea CREATE TABLE ... *111*
5.1.2. Instrucțiunile ALTER TABLE și DROP TABLE ... *113*
5.2. INSTRUCȚIUNILE DE ACTUALIZARE A BAZEI DE DATE ... 115
5.2.1. Instrucțiunea INSERT ... *115*
5.2.2. Instrucțiunea UPDATE ... *118*
5.2.3. Instrucțiunea DELETE ... *120*
5.3. GESTIUNEA INDECȘILOR ÎN MICROSOFT SQL SERVER MANAGEMENT STUDIO ... 121
5.3.1. Tipurile de indecși utilizați în SQL Server 2014 ... *121*
5.3.2. Crearea indecșilor asupra tabelelor existente ... *123*
5.3.3. Crearea indecșilor, pornind de la crearea tabelelor ... *126*
5.3.4. Modificarea indecșilor ... *126*
5.4. GESTIUNEA INDECȘILOR ÎN LIMBAJUL TRANSACT-SQL ... 127
5.4.1. Crearea indecșilor ... *127*
5.4.2. Modificarea indecșilor ... *128*
5.4.3. Vizualizarea indecșilor ... *129*

6. DIAGRAME, SCHEME ȘI SINONIME ... **131**

6.1. DIAGRAMELE ÎN BAZA DE DATE SQL SERVER ... 131
6.1.1. Crearea diagramelor ... *131*
6.1.2. Editarea diagramelor ... *132*
6.1.3. Modificarea formei de vizualizare a tabelelor ... *135*
6.2. ADMINISTRAREA SCHEMELOR ... 136
6.2.1. Crearea schemelor în mediul SQL Server Management Studio ... *137*
6.2.2. Crearea schemelor în limbajul Transact-SQL ... *138*
6.2.3. Modificarea schemelor ... *139*
6.2.4. Deplasarea obiectelor în scheme cu ajutorul SQL Management Studio ... *140*
6.2.5. Deplasarea obiectelor cu ajutorul limbajului Transact-SQL ... *140*
6.2.6. Interogarea schemelor ... *141*
6.2.7. Ștergerea schemelor ... *141*
6.3. CREAREA ȘI SUPRIMAREA SINONIMELOR ... 141
6.3.1. Ce este un sinonim? ... *141*
6.3.2. Crearea sinonimelor în SQL Management Studio ... *142*
6.3.3. Crearea sinonimelor în limbajul Transct-SQL ... *143*
6.3.4. Ștergerea sinonimelor ... *143*

7. ADMINISTRAREA VIZIUNILOR ... **145**

7.1. TABELE TEMPORARE ȘI VIZIUNI ... 145
7.2. ADMINISTRAREA VIZIUNILOR ÎN SQL SERVER MANAGEMENT STUDIO ... 146
7.2.1. Crearea viziunilor cu View Designer ... *146*
7.2.2. Salvarea viziunilor ... *149*
7.2.3. Excluderea unei coloane din setul de rezultate al viziunii ... *149*

7.2.4. Modificarea ordinii de prezentare a câmpurilor *150*
7.2.5. Crearea criteriilor în viziuni *150*
7.2.6. Acordarea permiselor *151*
7.2.7. Obținerea informațiilor despre proprietățile (dependențele) viziunilor *151*
7.2.8. Vizualizarea datelor din viziuni *153*
7.3. GESTIUNEA VIZIUNILOR, UTILIZÂND EDITORUL DE INTEROGĂRI 154
7.3.1. Sintaxa generală a instrucțiunii de creare a viziunilor *154*
7.3.2. Crearea viziunilor *155*
7.3.3. Operații DML asupra viziunilor *157*
7.3.4. Modificarea și suprimarea viziunilor *158*
7.4. EXPRESIE-TABEL 158
7.4.1. Caracteristici de bază ale CTE *159*
7.4.2. Interogări CTE nerecursive *159*
7.4.3. Eliminarea liniilor duplicate din tabele virtuale *161*
7.4.4. Interogările recursive folosind expresii-tabel *162*
7.4.5. O expresie-tabel recursivă pentru calculul factorialului unui număr *163*
7.4.6. O expresie-tabel recursivă pentru parcurgerea unei ierarhii *164*

8. PROCEDURI STOCATE ȘI FUNCȚII DEFINITE DE UTILIZATOR 167

8.1. NOȚIUNI DE ELABORARE A PROCEDURILOR STOCATE 167
8.1.1. Ce sunt procedurile stocate? *167*
8.1.2. Clasificarea procedurilor stocate *167*
8.1.3. Reguli de creare a procedurilor stocate *169*
8.1.4. Structura procedurilor stocate *169*
8.2. CREAREA PROCEDURILOR STOCATE 170
8.2.1. Crearea procedurilor stocate în Transact-SQL Query Editor *170*
8.2.2. Crearea procedurilor stocate din arborele de obiecte *171*
8.3. EXECUTAREA PROCEDURILOR STOCATE 172
8.3.1. Executarea procedurilor stocate în Transact-SQL Query Editor *172*
8.3.2. Executarea procedurilor stocate în SQL Server Management Studio *174*
8.4. MODIFICAREA PROCEDURILOR STOCATE 176
8.5. FUNCȚII DEFINITE DE UTILIZATOR 177
8.5.1. Funcții de tip scalar *177*
8.5.2. Funcții de tip tabel simplu *180*
8.5.3. Funcții de tip tabel multi-instrucțiune *181*
8.5.4. Modificarea și suprimarea funcțiilor definite de utilizator *182*
8.6. FUNCȚIILE DEFINITE DE UTILIZATOR VS PROCEDURI STOCATE 183

9. CREAREA ȘI UTILIZAREA DECLANȘATOARELOR 187

9.1. CLASIFICAREA TIPOLOGICĂ ȘI STRUCTURA DECLANȘATOARELOR 187
9.1.1. Declanșatoare DML *187*
9.1.2. Declanșatoare DDL *188*
9.1.3. Declanșatoare Logon *189*
9.1.4. Structura declanșatoarelor *189*
9.2. CREAREA, ACTIVAREA ȘI DEZACTIVAREA DECLANȘATOARELOR 193
9.2.1. Crearea declanșatoarelor DML *193*
9.2.2. Crearea declanșatoarelor DDL *195*
9.2.3. Crearea declanșatoarelor LOGON *195*

9.2.4. Activarea şi dezactivarea declanşatoarelor *196*
9.3. DECLANŞATOARELE ŞI PROCEDURILE STOCATE STANDARD 197
10. RECUPERAREA BAZEI DE DATE 199
10.1. MODELE DE RECUPERARE ŞI TIPURI DE BACKUPURI 199
10.1.1. Modele de recuperare *199*
10.1.2. Backupuri complete ale bazei de baze *200*
10.1.3. Backupuri diferenţiale *200*
10.1.4. Backupuri ale bazei de date *200*
10.1.5. Backupuri parţiale *201*
10.1.6. Backupuri de fişiere *201*
10.1.7. Backupuri ale jurnalului de tranzacţii (modelele complete şi Bulk-Logged) *201*
10.1.8. Backupuri copy-only *202*
10.2. RESTRICŢII ASUPRA OPERAŢIILOR DE RECUPERARE 202
10.2.1. Nu pot fi create copii ale datelor deconectate (offline) *202*
10.2.2. Restricţii asupra accesului concurenţial în timpul creării copiei de rezervă *202*
10.3. CREAREA COPIILOR DE REZERVĂ 203
10.3.1. Dispozitive backup *203*
10.3.2. Realizarea unui backup complet al bazei de date *205*
10.3.3. Vizualizarea copiei de rezervă *206*
10.3.4. Crearea backupului în Transact-SQL Query Editor *208*
10.4. RESTAURAREA BACKUPULUI 209
10.4.1. Recuperarea datelor în mediul SQL Server Management Studio *209*
10.4.2. Recuperarea datelor, folosind instrucţiuni Transact-SQL *210*
11. SERVICII DE INTEGRARE A DATELOR 213
11.1. COMPONENTELE ŞI FUNCŢIONALITĂŢILE SSIS 213
11.2. INSTALARE SQL SERVER DATA TOOLS (SSDT) 213
11.3. GESTIUNEA DATELOR, APLICÂND SSIS PACKAGE DESIGNER 217
11.3.1. Crearea şi exportarea pachetelor *217*
11.3.2. Depanarea pachetelor SSIS *224*
11.4. EXPORTAREA DATELOR CU AJUTORUL ASISTENTULUI 225
11.5. GESTIUNEA PACHETELOR SSIS ÎN MEDIUL SQL SERVER MANAGEMENT STUDIO 232
11.5.1. Importarea unui pachet SSIS *233*
11.5.2. Executarea unui pachet SSIS *235*
12. SERVICII DE CREARE A RAPOARTELOR 237
12.1. GESTIUNEA RAPOARTELOR CU AJUTORUL ASISTENTULUI 237
12.1.1. Crearea rapoartelor *237*
12.1.2. Previzualizarea rapoartelor *243*
12.1.3. Configurarea proprietăţilor rapoartelor *244*
12.1.4. Publicarea şi vizualizarea rapoartelor *246*
12.2. EDITAREA PROPRIETĂŢILOR UNUI RAPORT DEJA EXISTENT 247
13. SERVICII DE ANALIZĂ DIMENSIONALĂ A DATELOR 249
13.1. INTRODUCERE ÎN OLAP 249
13.1.1. Un exemplu de cub *249*
13.1.2. "Tăierea" cubului *250*

13.1.3. Metode de construire a cuburilor 251
13.2. CREAREA UNUI PROIECT ANALYSIS SERVICES 251
13.3. CREAREA UNUI CUB FOLOSIND ASISTENTUL, DAR FĂRĂ SURSA DE DATE 252
13.3.1. Construirea unui Cub nou 252
13.3.2. Specificarea metodei de construire 253
13.3.3. Adăugarea măsurilor 253
13.3.4. Adăugarea dimensiunilor 254
13.3.5. Definirea perioadelor de timp 255
13.3.6. Adăugarea calendarului special 256
13.3.7. Stabilirea legăturilor dintre dimensiuni şi măsuri 257
13.3.8. Finalizarea utilizării asistentului Cube Wizard 258
13.4. REVIZUALIZAREA STRUCTURII CUBULUI ÎN CUBE DESIGNER 259
13.5. CREAREA TABELELOR-DIMENSIUNI ŞI TABELELOR-FAPTE 261
13.5.1. Generarea unei Scheme folosind asistentul Schema Generation Wizard 261
13.5.2. Încărcarea datelor în schema relaţională 267
13.5.3. Procesarea dimensiunilor vs procesarea cuburilor 270
13.5.4. Desfăşurarea şi procesarea Cubului 271
13.6. INTEROGAREA CUBULUI, FOLOSIND LIMBAJUL MULTIDIMENSIONAL EXPRESSIONS 272
13.6.1. Limbajul Multidimensional Expressions 272
13.6.2. Executarea interogărilor MDX 272
13.6.3. Explorarea dimensiunilor şi măsurilor 274
BIBLIOGRAFIE **277**

INTRODUCERE

Microsoft SQL Server este un sistem de gestiune a bazelor de date relaţionale. Principalul limbaj de programare folosit de acest sistem este Transact SQL (T-SQL), care reprezintă o extensie a limbajului SQL creată de către Microsoft şi Sybase.

Codul de bază pentru Microsoft SQL Server îşi are originile în Sybase SQL Server şi a reprezentat intrarea Microsoftului pe piaţa bazelor de date pentru întreprinderi, concurând cu Oracle, IBM şi Sybase. Microsoft, Sybase şi Ashton-Tate s-au unit pentru a crea şi a scoate pe piaţă prima versiune numită SQL Server 4.2 pentru Win OS/2. Mai târziu, Microsoft a negociat pentru drepturi de exclusivitate la toate versiunile de SQL Server scrise pentru sistemele de operare Microsoft. Sybase şi-a schimbat, ulterior, numele în Adaptive Server Enterprise, pentru a evita confuzia cu Microsoft SQL Server.

SQL Server 7.0 a fost primul server de baze de date bazat pe GUI. O variantă de SQL Server 2000 a fost prima variantă comercială pentru arhitectura Intel. În ciuda amânărilor suferite de SQL Server 2005, acest produs a avut un succes foarte mare pe piaţa IT. Pe piaţă se observă, de asemenea, că interesul către SGBD-ul companiei Microsoft creşte.

SQL Server 2008, cunoscut şi sub numele de cod *„Katmai"*, a apărut în al doilea trimestru al anului 2008, sub formă de soft-test, însă cu valabilitate generală a apărut în trimestrul al treilea, la exact trei ani de la lansarea produsului SQL Server 2005, care, la rândul său, a suferit foarte multe amânări la lansare.

În cadrul evenimentului Pass Summit 2011, a fost oficial prezentată versiunea SQL Server 2012. Versiunea SQL server 2012 duce mai departe succesul înregistrat de SQL Server 2008 şi impresionează clienţii cu noi capabilităţi aflate în strânsă legătură cu noile trenduri IT. Acest sistem conţine o serie de idei şi concepte noi, care ulterior au stat la temelia formarii unei versiuni mai avansate – *SQL Server 2014*.

SGBD-ul SQL Server 2014 este disponibil începând cu 1 aprilie 2014 pe site-ul celor de la Microsoft. Există mai multe avantaje noi, care ar putea să motiveze companiile să mute bazele de date pe sistemul SQL Server 2014: performanţă îmbunătăţită, soluţii de business inteligence, procesare în memorie, disponibilitate mare şi recuperare în caz de dezastru, securitate şi conformitate, şi scalabilitate [9].

1. Motorul In-Memory OLTP

În SGBD-ul SQL Server 2014, a devenit posibilă optimizarea tabelelor selectate şi a procedurilor stocate cu scopul de a fi încărcate în memorie. Tehnologia In-Memory OLTP (aka Hekaton) este proiectat pentru tranzacţii de mare concurenţă şi utilizează un nou mecanism de control pentru eliminarea reţinerile cauzate de blocare. SQL Server 2014 oferă clienţilor acestui sistem o creştere de 20 de ori mai performantă, comparativ cu utilizării sistemului SQL Server 2012.

2. Îmbunătăţiri în funcţionarea AlwaysOn

Microsoft a îmbunătăţit tehnologia integrare AlwaysOn prin extinderea numărului maxim de replici secundare de la patru la opt. Replicile secundare accesibile pentru citire pot fi folosite pentru sarcinile de citire, chiar şi în cazul când replica primară este indisponibilă. În plus, SQL Server 2014, oferă un asistent nou - *Add Azure Replica Wizard*, care ar permite crearea replicilor secundare asincrone în Windows Azure.

3. Extinderea memoriei Buffer Pool

SQL Server 2014 prevede integrarea discurilor solide (SSD), fapt ce permit extinderea memoriei Buffer Pool în calitate de memorie nevolatilă rapidă, RAM (NvRAM). Oportunitatea extinderii memoriei tampon, cu ajutorul discurilor SSD, facilitează aplicarea sistemelor în care memoria este activată la maximum. Această opţiune îmbunătăţeşte performanţa aplicaţiilor OLTP, care sunt caracterizate prin lectură de mare intensitate.

4. Actualizarea indecşii Columnstore

Cu apariţia sistemului SQL Server 2012, pentru prima dată, au fost introduşi indecşii columnstore, fapt ce au accelerat simţitor interogările la depozitele de date. În unele cazuri, este posibilă creşterea de zece ori a productivităţii sistemului. Cu toate acestea, în SQL Server 2012, indecşii columnstore sunt utilizaţi doar în regimul "read-only". În SQL Server 2014, această restricţie este eliminată, datorită apariţiei indecşilor columnstore actualizabili. Dar indecşii columnstore trebuie să utilizeze toate coloanele din tabel şi nu pot fi combinaţi cu alţi indecşi.

5. Controlul I/O a magaziilor

Gestionarea resurselor limitează consumul CPU şi resursele de memorie pentru un anumit volum de muncă. În SQL Server 2014 poate fi extinsă raza de acţiune şi permisă gestiunea resurselor I/O a depozitelor. Astfel, controlul poate limita resursele fizice I/O, alocate pentru firele la nivel de utilizator în bazinul de resurse.

6. Power View pentru modele multidimensionale

Odată cu introducerea sistemului SQL Server 2014, tehnologia Power View, care, anterior, a funcţionat numai cu date tabelare, acum pot fi folosite şi cu modele multidimensionale (cuburi OLAP). Rapoartele Power View pot crea o varietate de date vizualizate, inclusiv tabele, matrice, diagrame cu bule şi hărţi geografice. Modelele Power View multidimensionale susţin interogări cu analize sofisticate folosind Data Analysis Expressions (DAX).

7. Power BI pentru Office 365 cu integrarea funcţiilor de business intelligence

Power BI pentru Office 365 este un cloud bazat pe Business Intelligence (BI), care oferă soluţii de navigare a datelor şi vizualizare a capabilităţii. Microsoft Power BI pentru Office 365 includ tehnologiile Power Query (fosta denumire - Data Explorer), Power Map (fosta denumire - GeoFlow), Power Pivot şi Power View.

8. SQL Server Data Tools pentru Business Intelligence

Noile instrumente de date SQL Server Data Tools pentru BI (SSDT-BI) sunt destinate rearării modelelor SQL Server Analysis Services (SSAS), rapoartelor SSRS şi pachetelor SSIS. Sistemele SSAS şi CSS sunt acceptate atât pentru SQL Server 2014, cât şi pentru versiunile anterioare, dar pentru crearea şi implementarea proiectelor pentru serviciile Integration Service (SSIS) sunt disponibile numai pentru sistemul SQL Server 2014. În versiunea preliminară a sistemului SQL Server 2014, programul SQL Server Setup nu prevedea instalarea SSDT- BI. Această componentă trebuie să fie descărcată separat la Microsoft Download Center.

9. Criptarea copiilor de rezervă (backup)

O inovaţie mult aşteptată a sistemului SQL Server 2014 a reprezentat-o dezvoltarea procedeelor de criptare a copiilor de rezervă a bazelor de date pentru a proteja datele stocate. SQL Server 2014 suportă mai mulţi algoritmi de criptare, inclusiv Advanced Encryption Standard (AES) 128, AES 192, AES 256 şi Triple DES. Criptarea backup-urilor a sistemului SQL Server 2014 necesită un certificat sau o cheie asimetrică.

10. Gestiunea copiilor de rezervă SQL Server în Windows Azure

În SQL Server 2014, au fost implementate funcţiile de backup integrare cu Windows Azure. Serverul local SQL Server 2014 şi exemplarele maşinilor virtuale pentru Windows Azure susţin copiile de rezervă ale magaziei Windows Azure, cu toate că nu suntem fericiţi de perspectiva depinderii de conexiune la internet pentru a efectua o restaurare a copiilor de rezervă. Posibilitatea creării copiilor de rezervă în Windows Azure este, de asemenea, complet inclusă şi în SQL Server Management Studio (SSMS).

1. INSTALAREA ŞI CONFIGURAREA SQL SERVER 2014

În acest capitol sunt tratate problemele cu privire la instalarea SGBD-ului MS SQL Server 2014, gestiunea serviciilor şi înregistrarea unui server, lucrul în mediul SQL Server Management Studio, precum şi administrarea securităţii şi a accesului la baze de date. În afară de aceasta, sunt examinate tehnologia client-server, componentele logice ale sistemului MS SQL Server 2014 şi principiile de gestiune şi partajare a accesului la date [4].

1.1. Sistemul SQL Server 2014

1.1.1. Baze de date client-server

Până la apariţia SGBD-urilor de tip client-server, stocarea datelor şi procesarea lor avea loc pe acelaşi calculator. După apariţia reţelelor locale, bazele de date au fost transferate pe calculatoarele centrale. Dacă un terminal solicita realizarea unei interogări asupra unui tabel din baza de date, întreg tabelul era transferat pe calculatorul terminal şi aici se efectua procesarea datelor. Dar dacă există mai multe calculatoare, care execută aceeaşi cerere, în acelaşi timp şi asupra aceluiaşi tabel? Evident, se va produce o creştere semnificativă a traficului în reţea, o scădere a vitezei de procesare şi, în consecinţă, vor apărea multe greşeli în sistem.

În bazele de date client-server, situaţia descrisă s-a schimbat considerabil şi, potrivit exemplului de mai sus, interogarea solicitată de terminal este procesată de server, iar la terminal se expediază numai răspunsul. Astfel, se reduce fluxul de date în reţea, creşte eficienţa sistemului, micşorându-se numărul de greşeli.

Un alt mecanism foarte important al sistemelor cu arhitectură client-server este cel de asigurare a securităţii, prin sistemele de evidenţă a utilizatorilor şi parolelor.

Pe piaţă, există mai multe SGBD-uri care susţin arhitectura client-server. Între ele, se înscriu Microsoft SQL Server, Oracle, PostgreSQL, MySQL, Sybase, Interbase etc.

1.1.2. Facilităţi noi oferite de SQL Server 2014

SQL Server a evoluat rapid de la un SGBD desktop mic la un sistem puternic orientat pe o performanţă şi scalabilitate înaltă. Deja, se cunoaşte faptul că sistemul SQL Server este un SGBD de tip client-server, iar odată cu apariţia versiunii noi a acestui sistem au apărut şi posibilităţi noi de creare şi gestiune a bazelor de date.

Microsoft SQL Server 2014, cea mai recentă platformă de management de date, aplicaţii şi infrastructura de la Microsoft, reprezintă noul trend în materie de Cloud OS şi deserveşte centrele de date ale clienţilor, cele ale furnizorilor de hosting şi cloudul public de la Microsoft. Printre beneficiile unei platforme consecvente se numără accesul uşor şi de oriunde la date, management, identificare şi virtualizare [10].

SQL Server 2014 le oferă organizaţiilor posibilitatea de a-şi scala, proteja şi accesa datele de pe desktop sau dispozitive mobile şi dintr-un centru de date sau din cloud (public, privat sau hibrid). SQL Server asigura performanţa mission-critical pentru cele mai solicitante aplicaţii de baze de date şi livrează în acelaşi timp cele mai performante soluţii de securitate, scalabilitate, accesibilitate şi suport. SQL Server 2014 îşi propune să livreze soluţii de business intelligence într-un mod accesibil cu ajutorul unor instrumente cunoscute, precum Microsoft Excel.

Microsoft SQL Server 2014 permite dezvoltarea de soluţii unice de cloud hibrid bazate pe Cloud OS. Aceste soluţii au un impact pozitiv asupra rezultatelor unei companii şi contribuie la dezvoltarea de soluţii inovatoare pentru aplicaţiile de baze de date.

Microsoft SQL Server 2014 este disponibil în următoarele ediţii:

- SQL Server 2014 Enterprise Edition
- SQL Server 2014 Business Intelligence Edition
- SQL Server 2014 Standard Edition

1.1.3. Componentele logice ale sistemului SQL Server

Datele, într-o bază de date SQL Server, sunt organizate în mai multe obiecte diferite. Microsoft de la o versiune la alta dezvoltă sistemul prin introducerea unor tipuri noi de obiecte care măresc flexibilitatea sistemului şi o fac mai robustă şi utilă. Aceste obiecte sunt ceea ce un utilizator poate să vadă când se conectează la o bază de date. Unele din tipuri de obiecte sunt specifice doar pentru sistemul SQL Server. Altele, sunt specifice oricărui SGBD relaţional. În continuare, este prezentată o scurta caracteristică a celor mai importante şi des utilizate componente ale unei baze de date.

Databases

Denumirea de bază de date denotă faptul că această componentă conţine obiecte responsabile de reprezentarea datelor într-o formă generală, precum şi accesul la acestea.

Tables

Tabelele, probabil, constituie partea principală a bazelor de date. Înregistrările sunt stocate sub formă de tabel (la nivel logic) constituit din linii şi coloane. Linia este formată dintr-o secvenţă de date, iar datele sunt reprezentate de coloane. Adică, fiecare coloană este responsabilă de păstrarea unui anumit tip de date, pe care le grupează după asemănare. Mulţimea de date dintr-o linie formează o înregistrare.

Database Diagrams

Diagramele bazei de date constituie o reprezentare grafică a tabelelor, indecşilor şi altor proprietăţi. Ele facilitează îndeplinirea unor sarcini, aplicând comenzi ale limbajului Transact–SQL. De exemplu, este mult mai uşoară executarea legăturii între două tabele, utilizând trasarea acesteia de la un punct la altul decât specificarea acestei legături prin instrucţiuni Transact–SQL.

Indexes

În bazele de date, indecşii sunt structuri adiţionale folosite pentru accelerarea accesului la date. Într-o bază de date, un index permite SGBD-ului să găsească, într-un tabel, datele necesare, fără a scana întregul tabel. Un index într-o bază de date este o listă de valori cu locaţiile de stocare a liniilor într-un tabel spre care indică fiecare valoare din listă. Indecşii pot fi creaţi atât pe o singură coloană, cât şi pe o mulţime de coloane dintr-un tabel şi pot fi implementaţi sub formă de B-arbori.

Views

O viziune nu este altceva decât un tabel virtual, rezultatul unei consultări. Datele accesibile direct într-o viziune nu sunt stocate în baza de date ca un obiect distinct. Ceea ce este stocat în baza de date este doar instrucţiunea *SELECT*. Setul de linii extrase de instrucţiunea *SELECT* formează acest tabel virtual, adică viziunea. Un utilizator poate folosi viziunea, indicând în instrucţiunile Transact-SQL numele viziunii în acelaşi mod în care se foloseşte numele unui tabel.

Schemes

Schema reprezintă un container de obiecte din baza de date unite printr-un nume (*Namespace*). Acest container se utilizează pentru denumirea obiectelor şi grupării lor logice şi pentru alocarea şi controlul accesului asupra acestor obiecte. La crearea unui nou utilizator, schema implicită a lui este ***dbo***. E posibilă specificarea unei scheme inexistente. Nu se permite specificarea unei scheme implicite pentru utilizator, care este, deja, asociată cu un grup Windows, cu un certificat sau o cheie asimetrică.

Synonyms

Un sinonim reprezintă un nume alternativ dat unui obiect al schemei bazei de date. Aplicațiile-client pot utiliza pentru referința la obiect numai o parte a numelui, dacă această parte a numelui reprezintă un sinonim al unui nume real, mai complex, dat obiectului. Acest instrument asigură un nivel de abstracție care protejează aplicația clientului de schimbările ce pot surveni în numele sau locația obiectelor bazei de date. Un sinonim aparține schemei şi, ca orice alt obiect al bazei de date, trebuie să aibă un nume unic.

Stored Procedures

O procedură stocată (*Stored Procedure*) reprezintă un grup de instrucțiuni Transact-SQL compilate într-un singur plan de execuție. Procedurile stocate SQL Server returnează date de patru tipuri:

- Parametri de ieşire, care returnează fie date (întregi sau secvențe de caractere), fie variabile-cursor.
- Coduri de returnare, care sunt întotdeauna valori întregi.
- Un set de rezultate pentru fiecare instrucțiune *SELECT* din procedura stocată sau din alte proceduri stocate apelate de procedura dată
- Un cursor global care poate fi indicat din exteriorul procedurii stocate.

Triggers

Declanşatoarele (*Triggers*) formează o categorie specială de proceduri stocate, care se execută automat atunci când o instrucțiune DML sau DDL este executată asupra tabelelor sau (in cazul declanşatoarelor DDL) chiar şi asupra serverului întreg. Declanşatoarele pot extinde logica de verificare a constrângerilor de integritate a bazei de date, valorilor implicite şi regulilor. Asemenea procedurilor stocate, declanşatoarele conțin instrucțiuni Transact-SQL.

Funcții

SQL Server 2014 oferă o gamă de funcții incluse (*built-in*), care permit executarea anumitor operații de procesare a datelor. Există posibilitatea de creare a funcțiilor definite de utilizator. De regulă, funcțiile sunt specificate de parametri, însă există şi excepții – funcții ce nu au parametri (aceste funcții, de obicei, se află alături de cuvântul *DEFAULT*). Parametrii care specifică baza de date, calculatorul, *loginul* sunt, deseori, opționali. Dacă aceştia nu sunt specificați, implicit este utilizată baza de date curentă, calculatorul-gazdă etc. SQL Server 2014 distinge 4 categorii de funcții. Cu cele mai importante se va face cunoştință în procesul studierii limbajului Transact-SQL.

Tipuri de Date

În SQL Server, fiecare coloană, variabilă locală, expresie este asociată cu un tip de date. SQL Server oferă o gamă foarte largă de tipuri de date. Cu toate acestea, în Transact-SQL sau Microsoft .NET Framework, utilizatorul poate crea tipurile sale de date. Tipurile de date alternative (definite de utilizator) se bazează pe tipurile de date din sistem. Caracteristicile acestor tipuri de date preiau din metodele şi operatorii claselor create în limbajele susținute de Framework.

Reguli

Regulile sunt instrumente care îndeplinesc aceleaşi funcții ca şi constrângerile *CHECK*. Utilizarea constrângerilor *CHECK* este mai preferată pentru restricționarea valorilor coloanelor. Acestea, de asemenea, sunt mai concise decât regulile. Asupra unei coloane poate fi aplicată doar o singură regulă, spre deosebire de constrângerea *CHECK* care poate avea o aplicare multiplă asupra unei coloane. Constrângerile *CHECK* sunt parte componentă a instrucțiunii *CREATE TABLE*, în timp ce regulile sunt create ca obiecte aparte şi apoi sunt legate de coloane.

Sequences

O secvență este un obiect din schema definit de utilizator. Rolul unei secvenței este destinat generării valorilor numerice, conform specificațiilor definite la crearea secvenței. Secvența de valori numerice este generată, într-o ordine crescătoare sau descrescătoare, la un interval definit. Secvențele, spre deosebire de coloanele de tip ***Identity***, nu sunt asociate cu tabelele. Aplicațiile de utilizare pot referi un obiect de secvență şi coordona valorile cheilor în mai multe rânduri şi tabele.

1.2. Instalarea şi configurarea sistemului

Pentru instalarea sistemului SQL Server 2014, se parcurg trei etate consecutive [4]:

1. Controlul necesităților hard şi software, controlul configurației sistemului, controlul setărilor de securitate.
2. Derularea procesului de instalare a sistemului SQL Server.
3. Utilizarea instrumentelor SQL Server pentru configurarea SGBD-ului.

1.2.1. Capacitățile hardware şi software necesare

Realizarea instalării necesită, în primul rând, cunoştințe cu privire resursele hardware. Necesitățile minime pentru SQL Server 2014 diferă în funcție de versiunea aleasă. În tabelul de mai jos, se pot vedea cerințele generalizate pentru SGBD-ul SQL Server 2014:

Hardware	Configurație minimă
Procesor	*Frecvența minimă:* Procesor x86: 1.0 GHz; Procesor x64: 1.4 GHz *Tip procesor:* Procesor x64: AMD Opteron, AMD Athlon 64, Intel Xeon cu Intel EM64T, Intel Pentium IV cu EM64T Procesor x86: Pentium III-compatibil sau mai rapid
Memorie RAM	Minimum 1GB (4GB sau mai mult, este recomandat), pentru *Express Edition* minimum recomandat este 512Mb.
Disc Drive	DVD ROM Drive pentru instalare de la disc optic
Spațiu pe discul rigid	În funcție de configurarea aleasă, minimal necesită 6 GB
Monitor	Rezoluție Super - VGA 800x600 sau mai mare

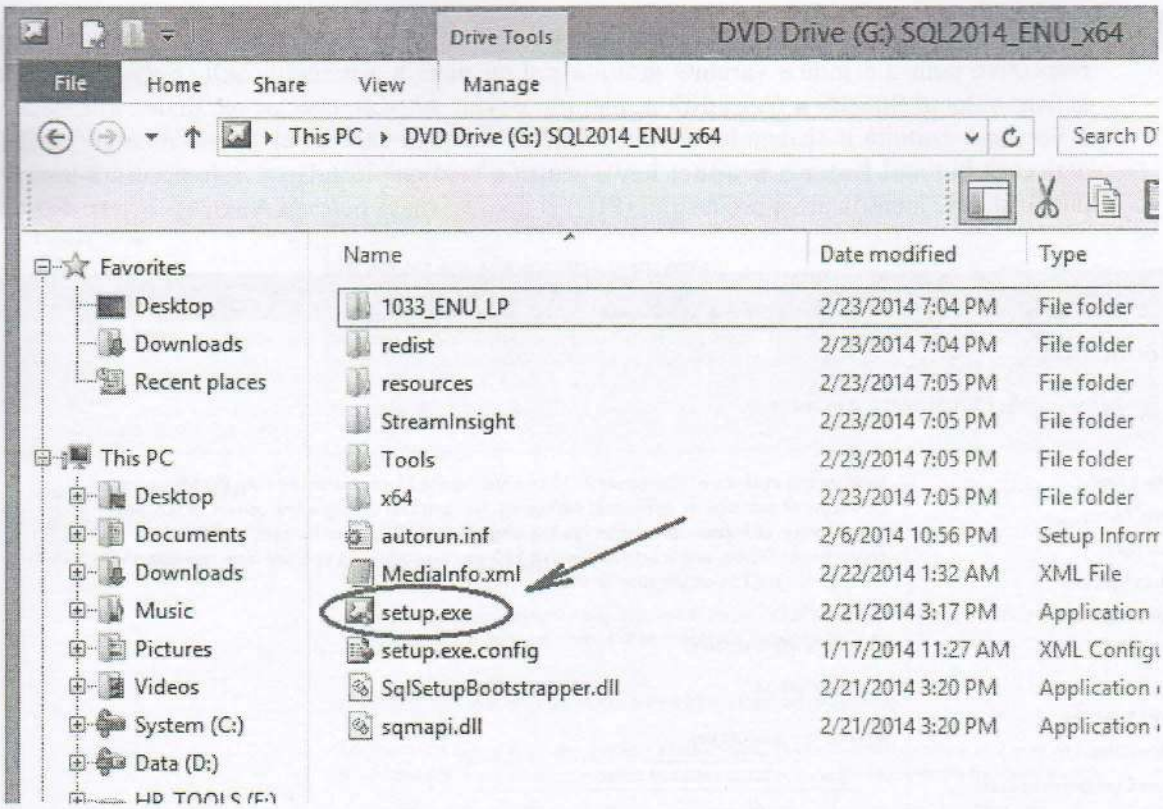
În următorul tabel, sunt prezentate cerințele soft generalizate pentru sistemul SQL Server 2014:

Software	Exigențe
Sistem de operare	Windows 7 SP1 sau mai superior – Windows Server 2008 SP2 sau mai superior
Framework	.NET Framework 3.5 SP1
Internet Explorer	Versiunea 7 sau mai superioară
Windows PowerShell	Windows PowerShell 2.0

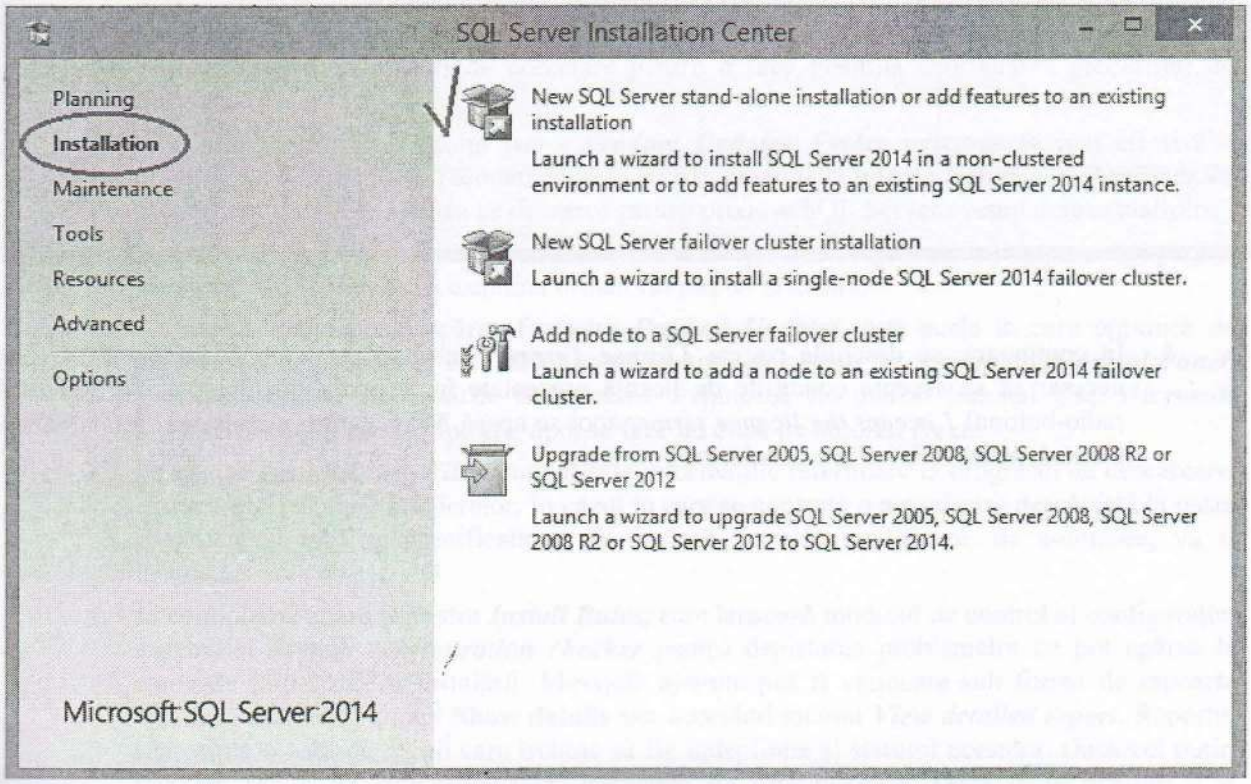
1.2.2. Instalarea sistemului

Procesul de instalare a SGBD-ului SQL Server 2014 se poate deosebi un pic în funcție de ediție şi componente selectate spre instalare. Mai jos se descrie procesul de instalare a sistemului SQL Server 2014 Development Edition x64, utilizând asistentul.

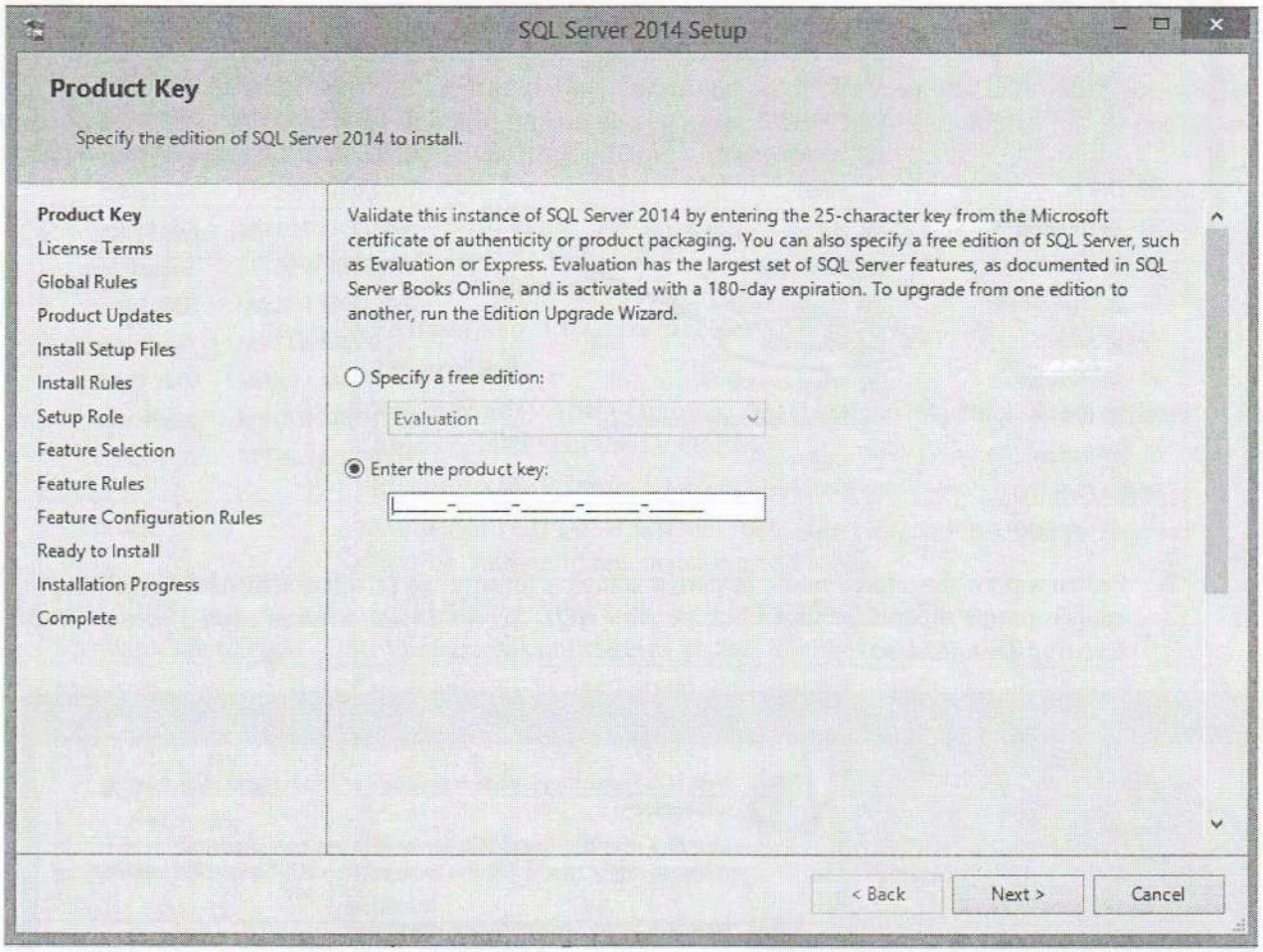
1. Pentru a porni instalarea, se accesează kit-ul de instalare, şi se aplică un dublu-click pe fişierul ***Setup.exe,*** asistentul de instalare lansează ***SQL Server Installation Center.***



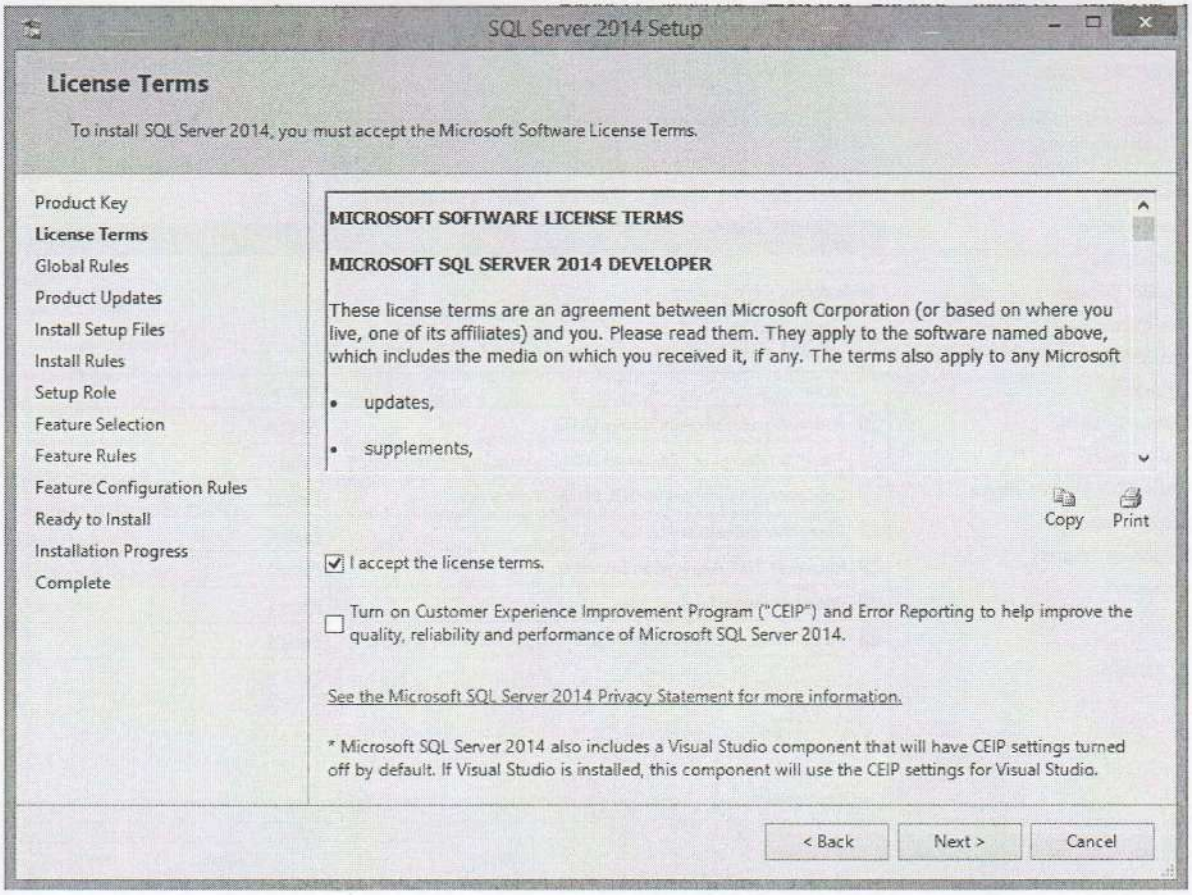
2. Pentru a porni instalarea nouă, în partea stângă a ferestrei se selectează meniul ***Installation*** şi apoi în partea dreaptă se face click pe ***New SQL Server Installation or Add Features to an Existing Installation.***



3. Prima fereastră a asistentului de instalare este ***Product key.*** Se activează radio-butoanele respective pentru a indica varianta gratuită sau cu plată a sistemului SQL Server. Dacă este activat butonul **Specify a free edition**, mai jos, devine activ un check-box, unde se poate alege o versiune gratuită a sistemului SQL Server. În cazul în care se alege varianta cu plată (se activează butonul **Enter a product key**), pentru a continua instalarea, este necesară inserarea numărului de identificare a produsului (PID) și apoi se apasă butonul **Next**.



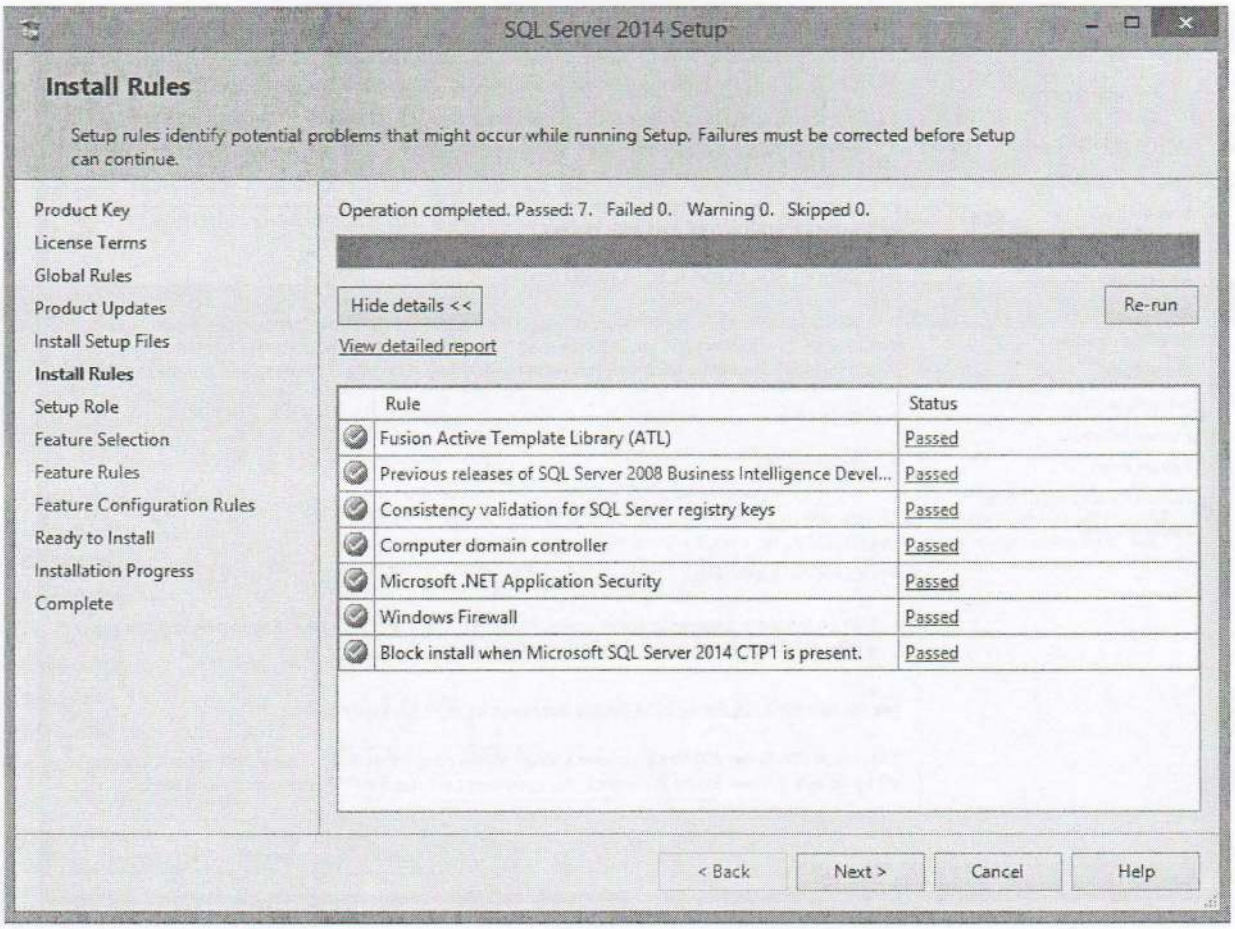
4. În continuare, se deschide pagina ***License Terms.*** Pentru a instala SQL Server 2014, este necesar să se accepte condițiile de licență prezentate în fereastra din dreapta. Se activează radio-butonul ***I accept the licence terms,*** apoi se apasă **Next**, pentru a continua instalarea sau **Cancel**, pentru a întrerupe instalarea.



5. Mai departe, la pasul ***Global Rules,*** asistentul va parcurge mai multe verificări şi va trece, în mod automat, la pasul următor, dacă nu vor fi depistate erori. În caz de eşec, la verificare, asistentul va livra măsuri de corectare pentru a face posibilă continuarea procesului de instalare.
6. Asistentul trece la următorul pas - ***Product Updates***. Pentru parcurgerea mai efectivă a acestui pas, Microsoft vă recomandă să aveţi o conexiune la internet activată. Conexiunea va oferi sistemului posibilitatea să descarce pentru produse SQL Server eventualele actualizări.

   În cazul în care nu sunt descoperite actualizări ale produselor, asistentul va parcurge pagina aceasta fără a o afişa şi va examina următorul pas de instalare.

   Un alt caz când poate apărea fereastra ***Product Updates,*** este acela în care opţiunea de verificare a actualizărilor este deconectată la nivel de sistem de operare din ***Control Panel***. Pentru a include actualizările la instalare, trebuie să fie marcat butonul **Use Microsoft Update to check for an update**, apoi se face un click pe butonul **Next**.
7. Pe pagina ***Install Setup Files,*** sunt afişate informaţiile referitoare la progresul de descărcare, extragere şi instalare a fişierelor. În cazul în care se constată o actualizare descărcată la pasul precedent şi ea este specificată să fie inclusă, această modificare, de asemenea, va fi instalată.
8. În continuare, apare fereastra ***Install Rules,*** care lansează modulul de control al configuraţiei sistemului ***System configuration checker*** pentru depistarea problemelor ce pot apărea la etapa de iniţializare a instalării. Mesajele apărute pot fi vizionate sub formă de rapoarte detaliate, apăsând butonul **Show details** sau accesând meniul ***View detailed report***. Raportul reprezintă o listă de reguli care trebuie să fie îndeplinite şi statutul acestora. Dacă cel puţin una din reguli nu este respectată, instalarea nu este posibilă. Dacă toate regulile sunt respectate (statutul este ***Passed***), asistentul trece la etapa următoare în mod automat . Dacă vreo regulă are statutul ***Failed,*** atunci instalarea se anulează până la corectarea cauzei ce a provocat eşecul, adică se apasă butonul **Cancel**

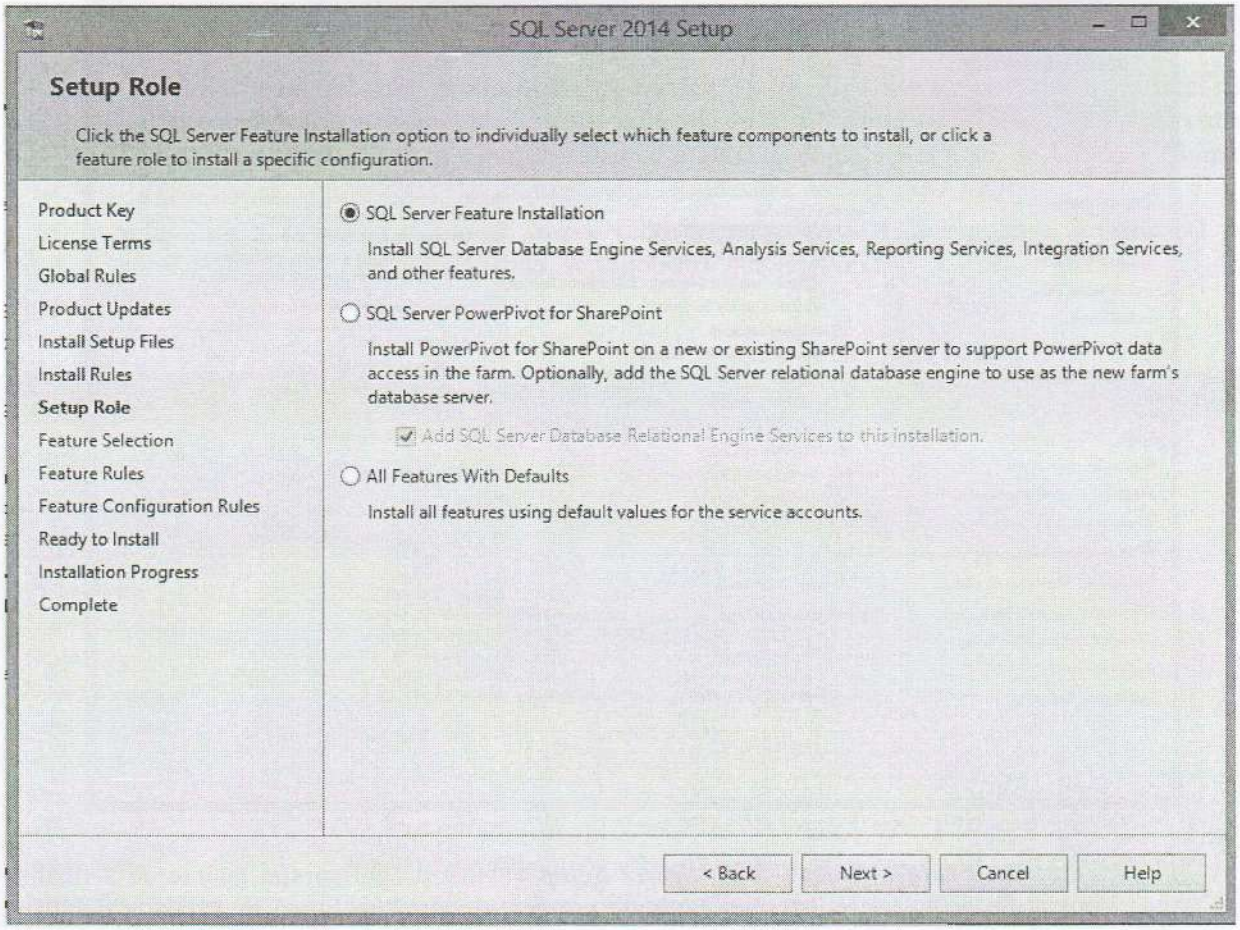


9 **Setup Role**

Pentru instalare, sunt disponibile 3 opțiuni.

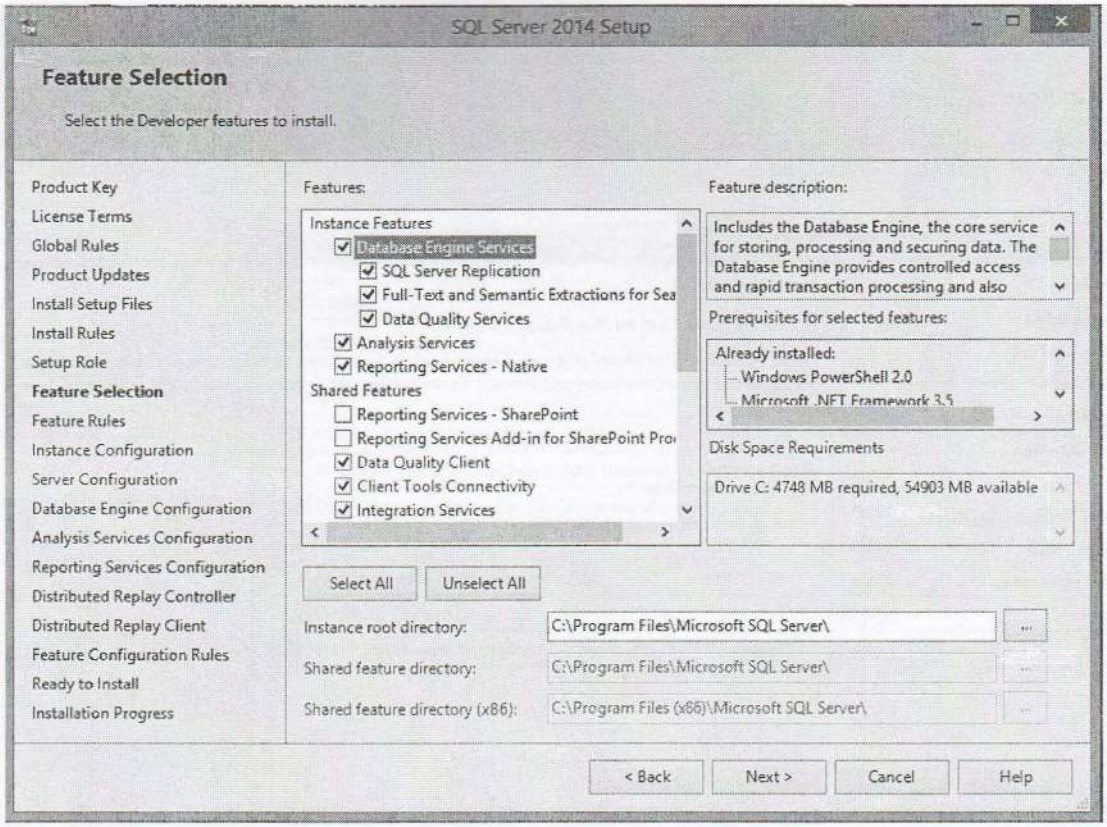
- Se alege opțiunea ***SQL Server Feature Installation***, dacă se doreşte instalarea componentelor SQL Server native: ***Database Engine Services***, ***Analysis Services*** şi ***Reporting Services***.
- Opțiunea ***SQL Server PowerPivot for SharePoint*** se alege pentru a instala in ***SharePoint*** componentele ***Analysis Services*** şi ***PowerPivot System Service*** care ulterior vor fi utilizate pentru gestiunea tabelelor Pivot din Excel.
- Opțiunea ***All Features With Defaults*** instalează toate caracteristicile disponibile cu setările implicite. Setul implicit exclude serviciul ***PowerPivot*** din ***SharePoint***.

În cazul nostru, se alege ***SQL Server Feature Installation*** sau ***All Features With Defaults***, apoi se apasă **Next**, pentru a continua instalarea sau **Cancel** pentru a întrerupe instalarea.

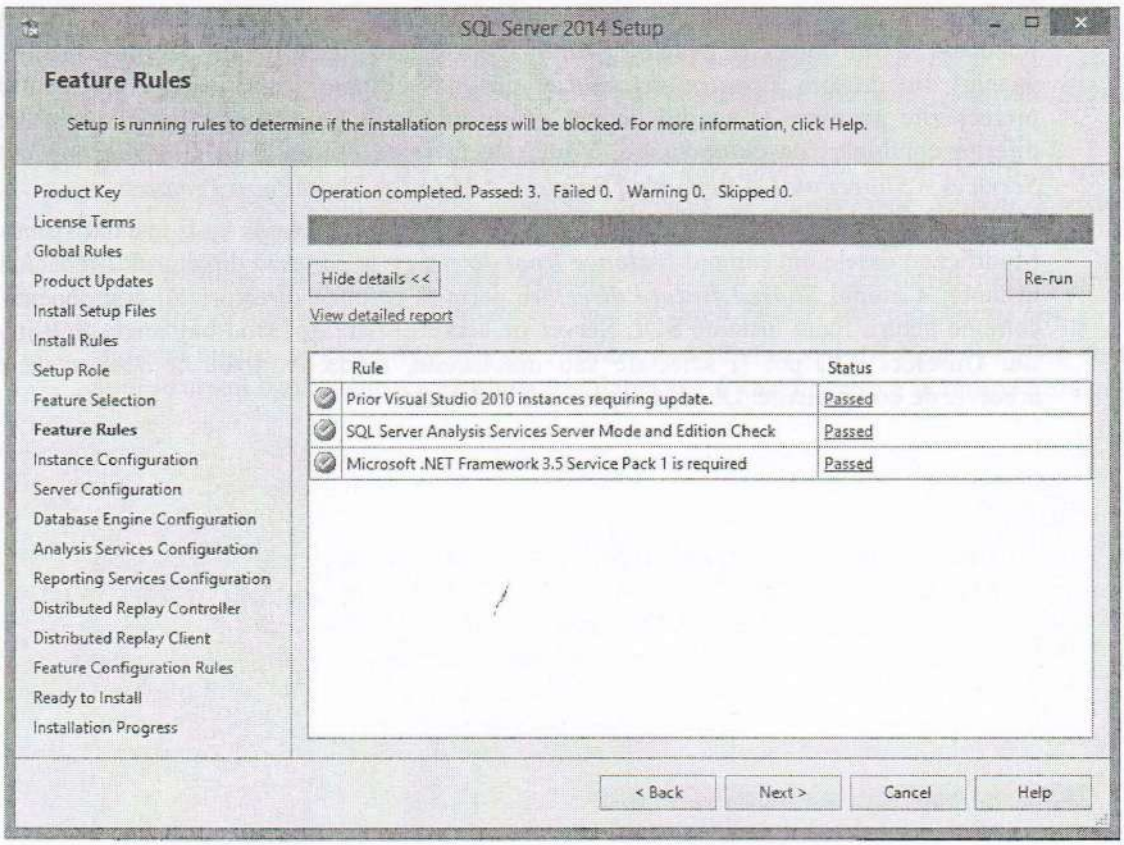


10. În fereastra ***Feature Selection***, se aleg componentele SGDB-ului ce vor fi instalate, selectând vreun obiect în panelul ***Features*** din centru din arborele de componente. În panelul din dreapta ***Feature description*** poate fi citită o scurtă descriere a acestuia, prerechizite necesare şi spaţiul ocupat de discul rigid. La instalare pot fi programate diferite combinări de componente. Vom selecta toate opţiunile în afară de ***Reporting Services – SharePoint*** şi ***Reporting Services Add-in for SharePoint Products***.

    Tot în această fereastră se poate modifica adresa directoriului unde va fi instalat sistemul. Modificând datele din câmpul ***Instance Root directory*** se editează directorului-rădăcină al instanţei. Câmpul ***Shared feature directory*** permite editarea directoriului componentelor comune pentru toate instanţe SQL Server pe serverul dat. Apăsând butoanele **Select All** sau **Unselect All,** pot fi selectate sau deselectate, respectiv, toate componentele din arborele de componente. După, se apasă butonul **Next**.



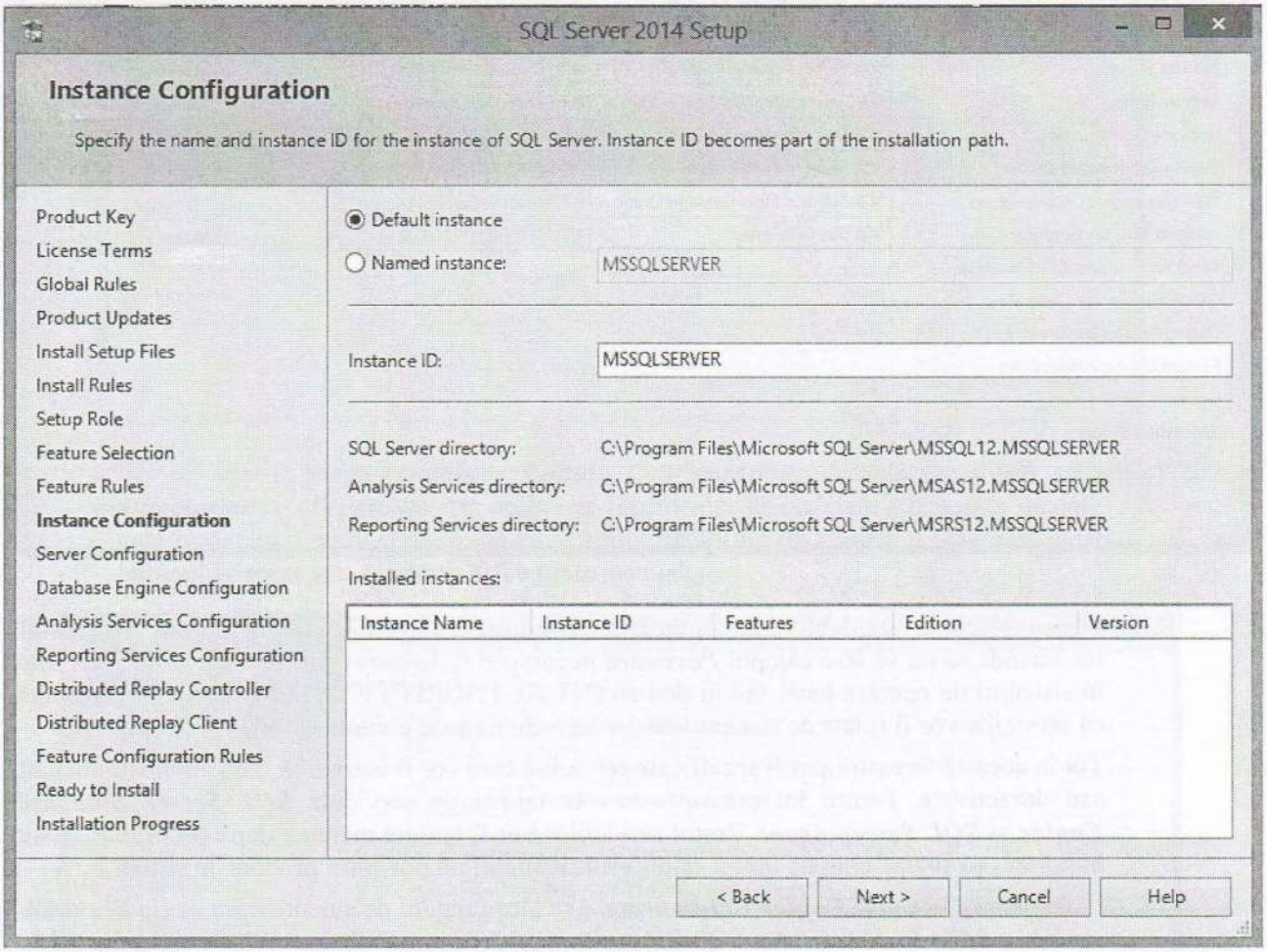
11. Fereastra ***Feature Rules - SQL Server Setup*** validează configurația înainte de a finaliza operațiile de instalare. Mesajele apărute pot fi vizionate sub formă de rapoarte detaliate, apăsând butonul **Show details** sau accesând meniul ***View detailed report***. Activitățile aplicate în această fereastră sunt similare cu cele din pasul 8.



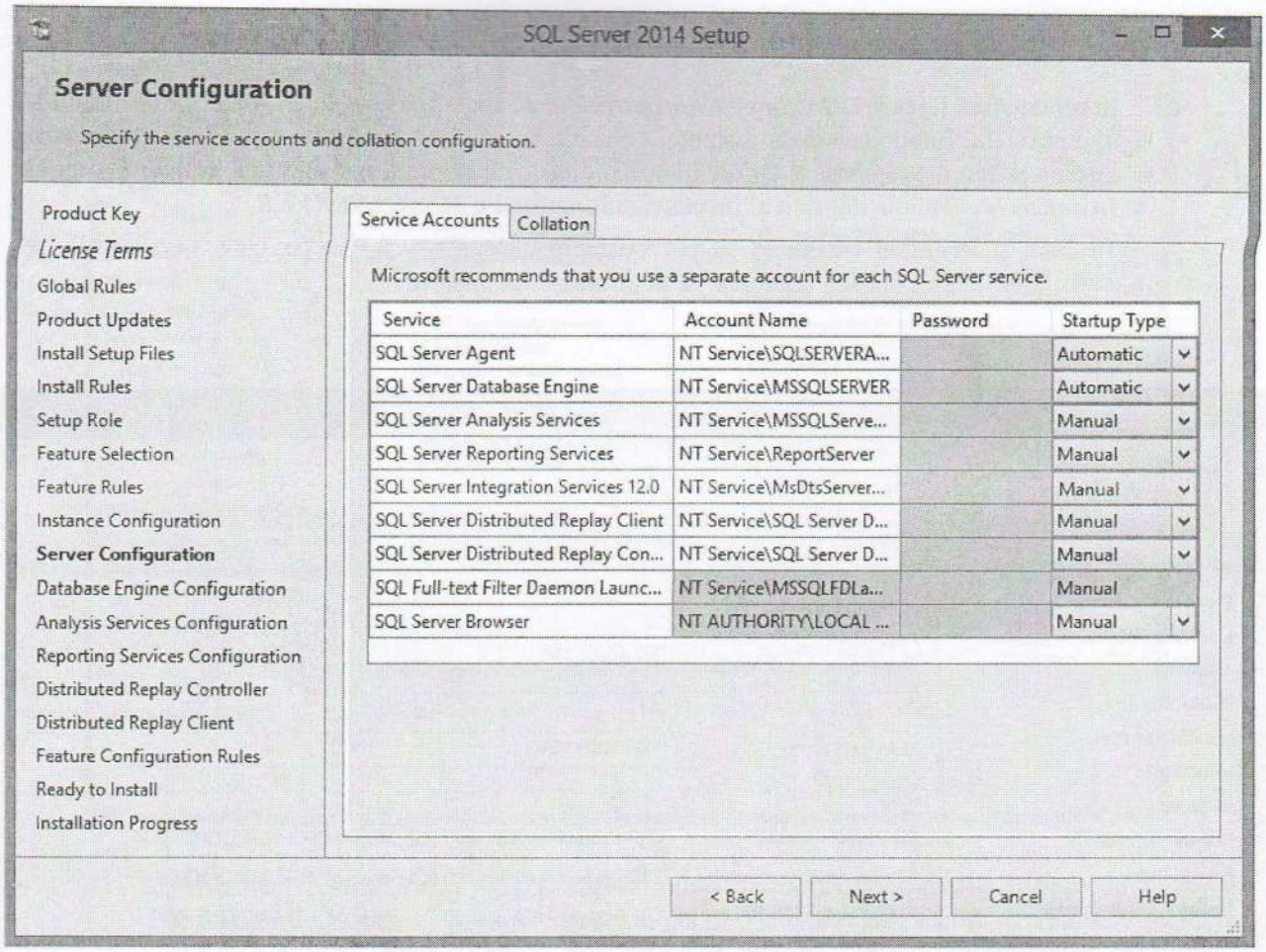
1.2.3. Configurarea sistemului

12. În următoarea fereastră, ***Instance Configuration,*** se specifică numele instanței sau sufixul Id-ului acesteia. Implicit numele instanței coincide cu Id-ul. Acest lucru este specific și pentru instanțele numite. Pentru a da un Id-sufix diferit de numele instanței, se editează câmpul ***Instance Id***. Numele implicit al instanței și Id-sufixul e ***MSSQLSERVER***.

    În câmpul ***Installed instances*** se pot vedea instanțe ale SQL Server care, deja, există pe calculatorul dat. Pentru a continua, se acționează butonul **Next**.



13. În fereastra ***Server Configuration*** a asistentului de instalare, este necesar să se atribuie serviciile SQL Server pe conturile utilizatorilor din numele cărora vor rula serviciile respective. Se poate atribui un cont pentru toate serviciile sau se poate configura fiecare serviciu individual.

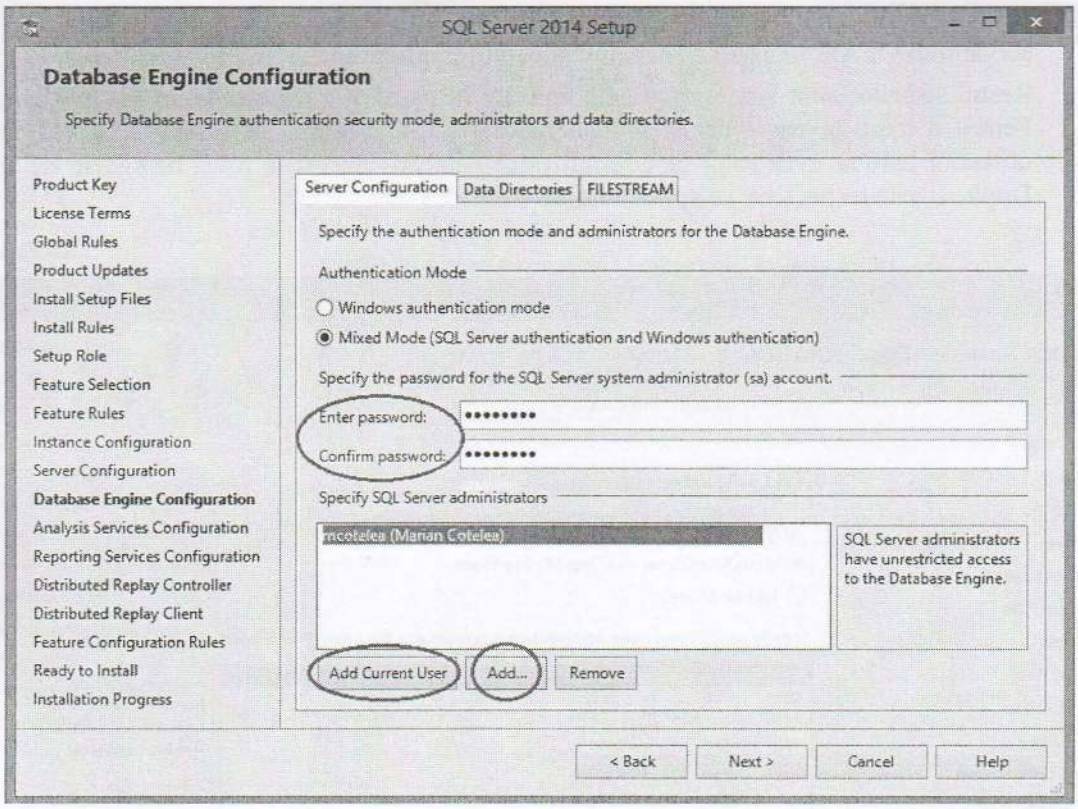


Pentru alocarea serviciilor pe conturi, se populează câmpul ***Account Name***. Microsoft recomandă să nu se lase câmpul ***Password*** necompletat. În cazul dat, se aleg conturi definite în sistemul de operare local sau în domen (NT AUTHORITY\SYSTEM). Aceasta înseamnă că serviciile vor fi rulate de sistemul de operare din numele contului local.

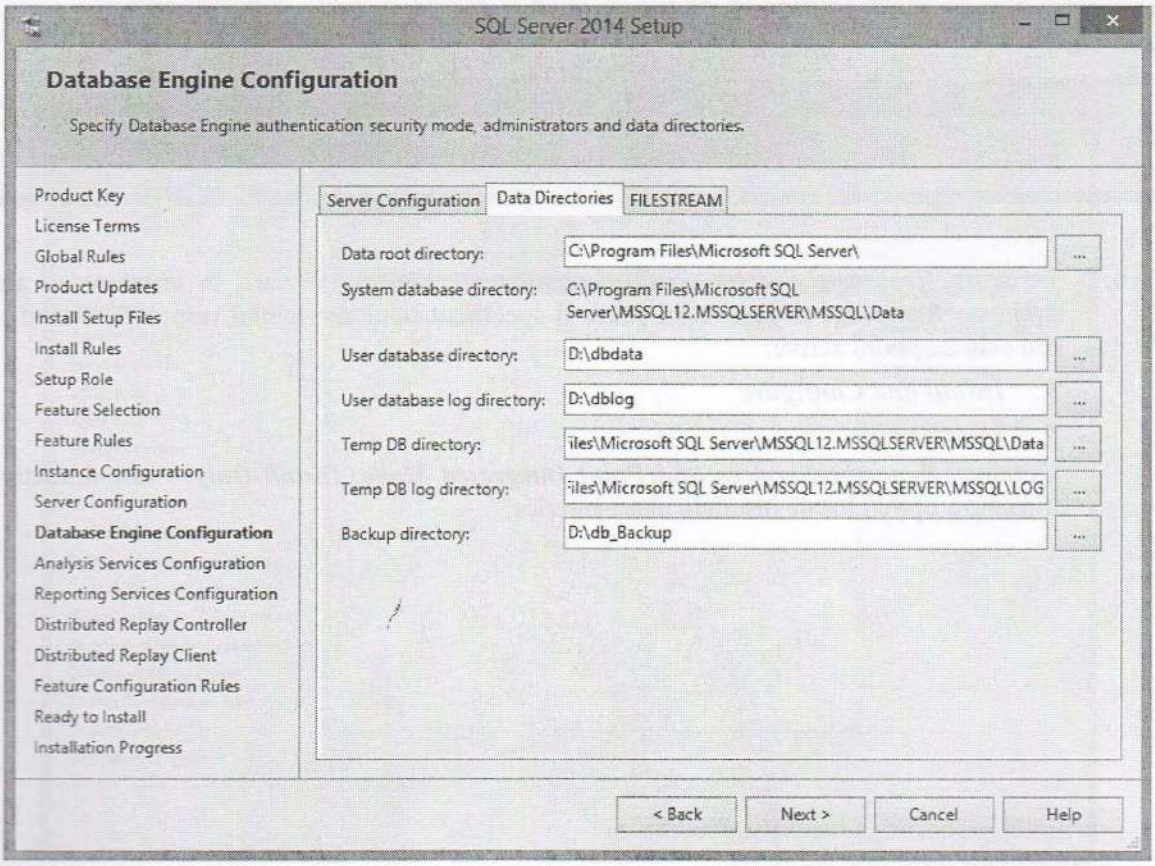
Tot în această fereastră pot fi specificate serviciile care vor fi lansate în mod automat, manual sau dezactivate. Pentru început, este nevoie numai de serviciile ***SQL Server Database Engine*** şi ***SQL Server Agent***. Restul serviciilor vor fi lansate manual, după necesitate. (Este important să se ţină cont de faptul că utilizatorii simpli nu pot lansa procese de sistem.)

14. În fereastra ***Database Engine Configuration*** se alege modul de autentificare de tip Windows sau de tip Mixt (autentificarea SQL Server împreună cu autentificarea Windows). În cazul în care se alege modul Mixt de autentificare, automat se activează login-ul de *SQL Server system administrator* (***sa***), care este predefinit în sistem. Pentru acest cont, este necesar să se creeze parola de acces (câmpurile ***Enter password*** şi ***Confirm password***) .

    Pentru a trece la tab-ul următor trebuie specificat cel puţin un utilizator care ar avea rol de SQL Server administrator de tip NT. Daca se doreşte ca acest rol să fie atribuit utilizatorului curent, se apasă butonul **Add Current User**. Pentru a alege alt cont de administrare a sistemului SQL Server, se apasă butonul **Add**. Dacă se doreşte să se excludă vreun cont din lista administratorilor, se apasă **Remove**.

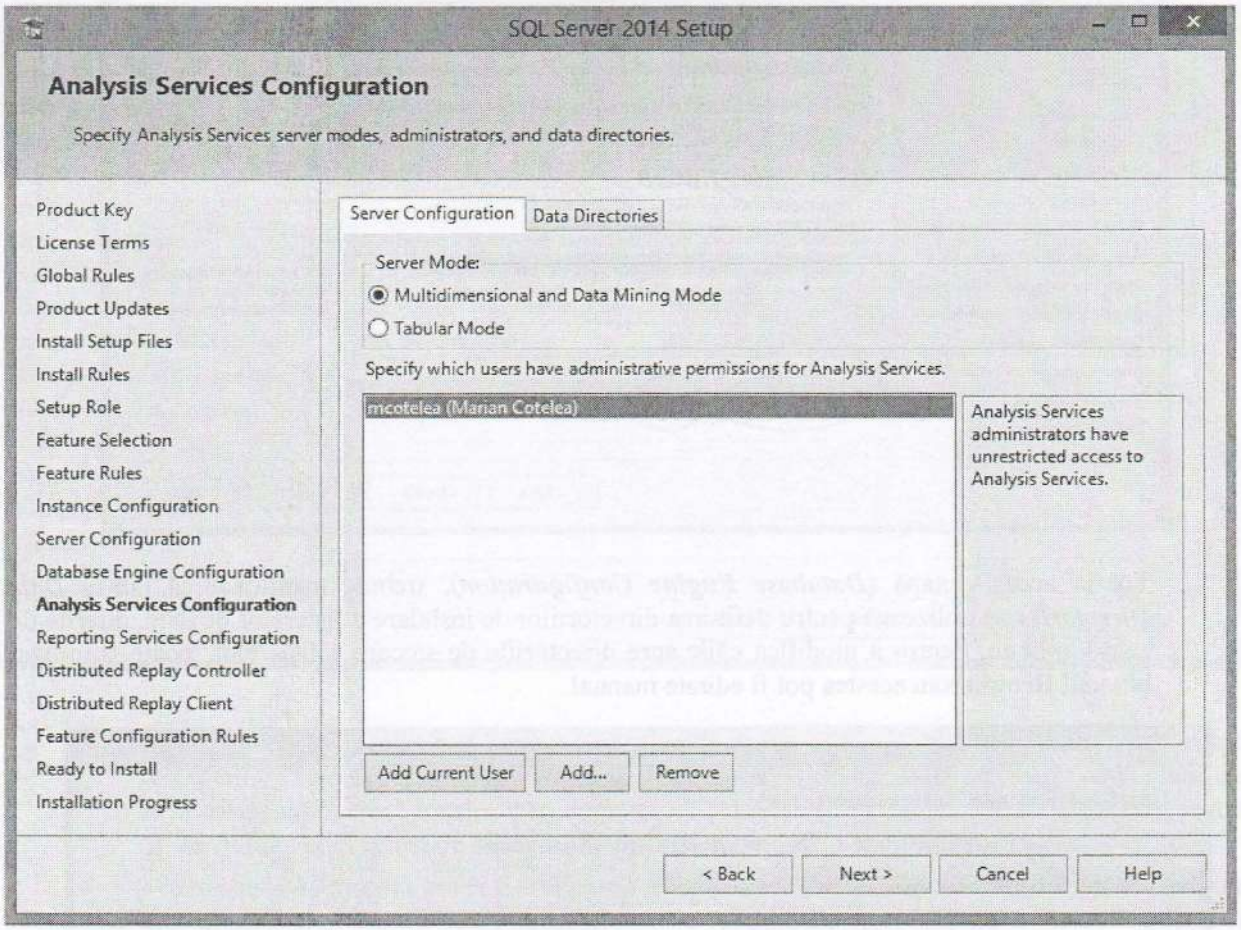


Tot la această etapă (***Database Engine Configuration)***, trebuie menționat că tab-ul ***Data Directories*** se utilizează pentru definirea directoriilor de instalare a fișierelor de date, diferite de cele implicite. Pentru a modifica căile spre directoriile de stocare a fișierelor, poate fi utilizat butonul **Browse** sau acestea pot fi editate manual.



15. În fereastra ***Analysis Server Configuration.*** apărută se execută operațiuni de configurare a serviciului OLAP. In cazul nostru, rămâne totul implicit.

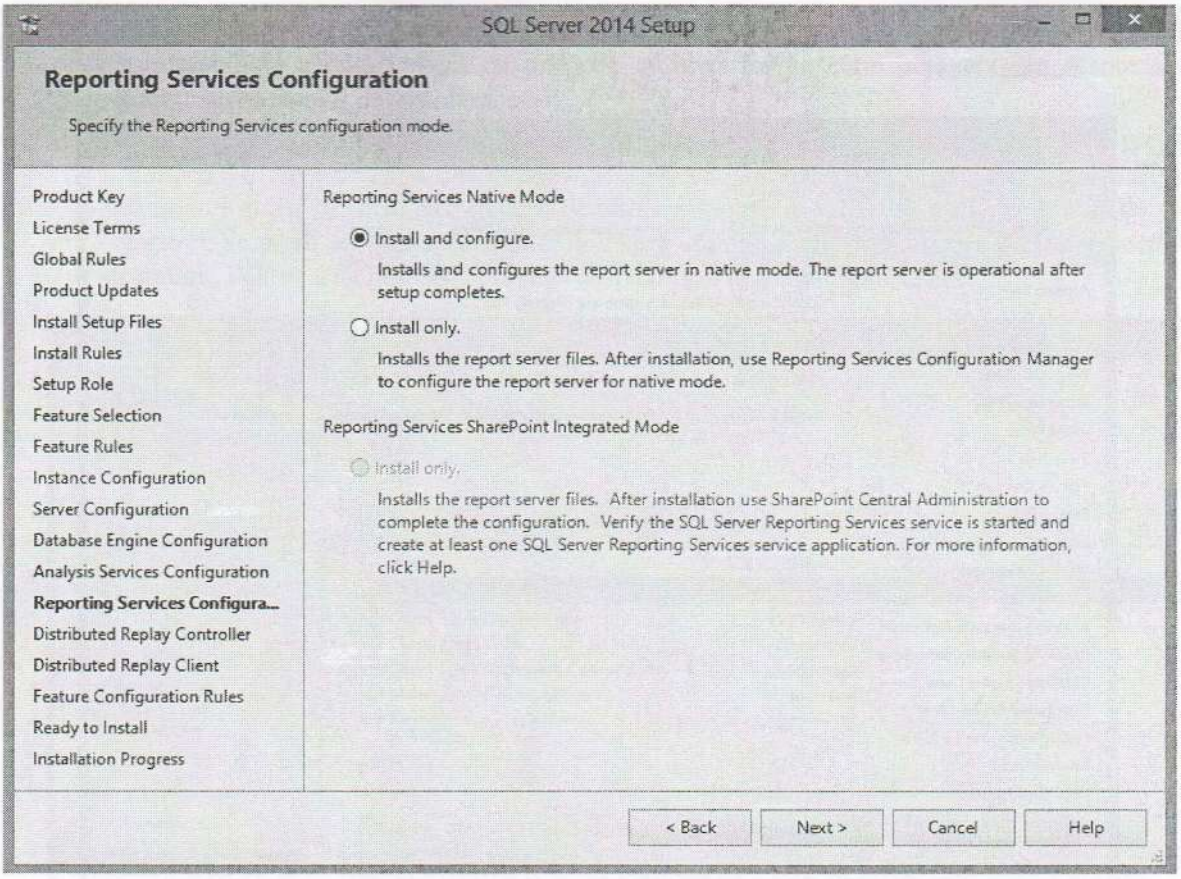
    Restul setărilor sunt similare cu cele aplicate la pasul 14, ***Database engine configuration.*** Pentru a trece la etapa următoare de instalare, este necesar să se specifice cel puțin un utilizator care ar avea rolul de administrator Analysis Server la nivel de sistem de operare. După setările respective, se apasă butonul **Next**.



16. Fereastra ***Reporting Service Configuration*** apare în cazul în care, la instalare, se alege opțiunea ***Reporingt service***. Aici poate fi specificat tipul serviciului respectiv, bifând una din cele 2 opțiuni active:

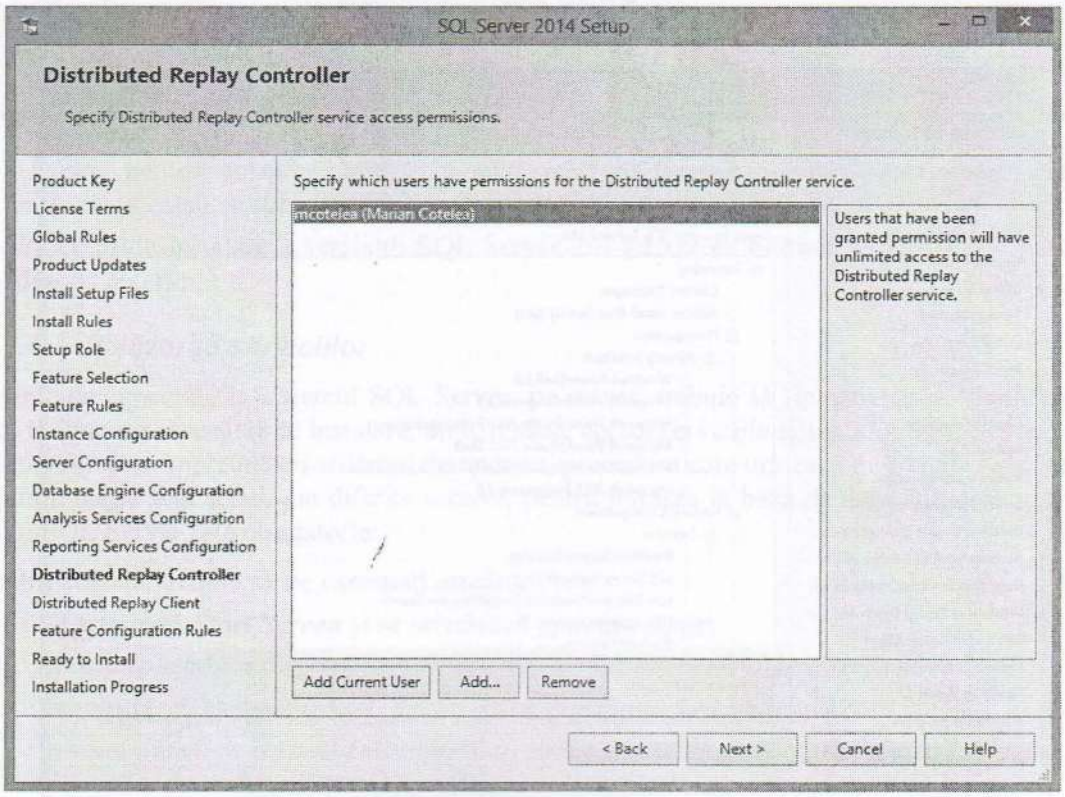
    - ***Install and Configure***
    - ***Install only***

    Opțiunea ***ReportingServices SharPoint Integrated Mode, Install Only*** – este dezactivată din cauza opțiunilor de instalare alese anterior.

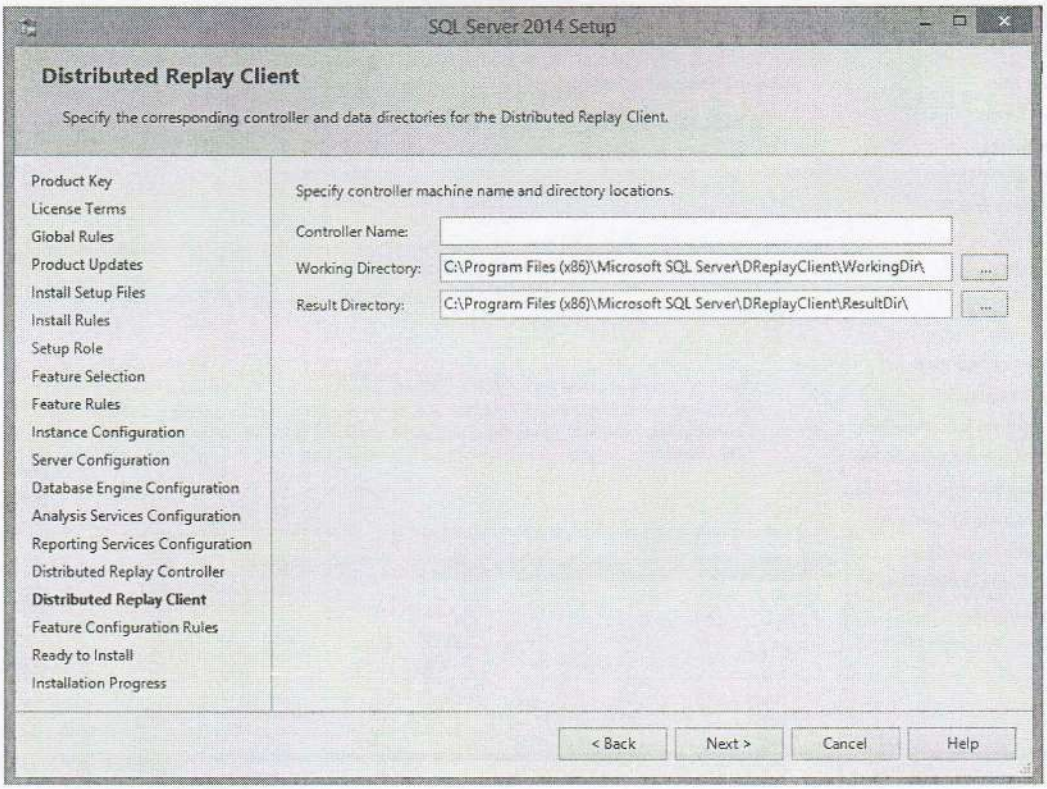


Se va lăsa configurația implicită (***Install and configure***) și se apăsă butonul **Next**.

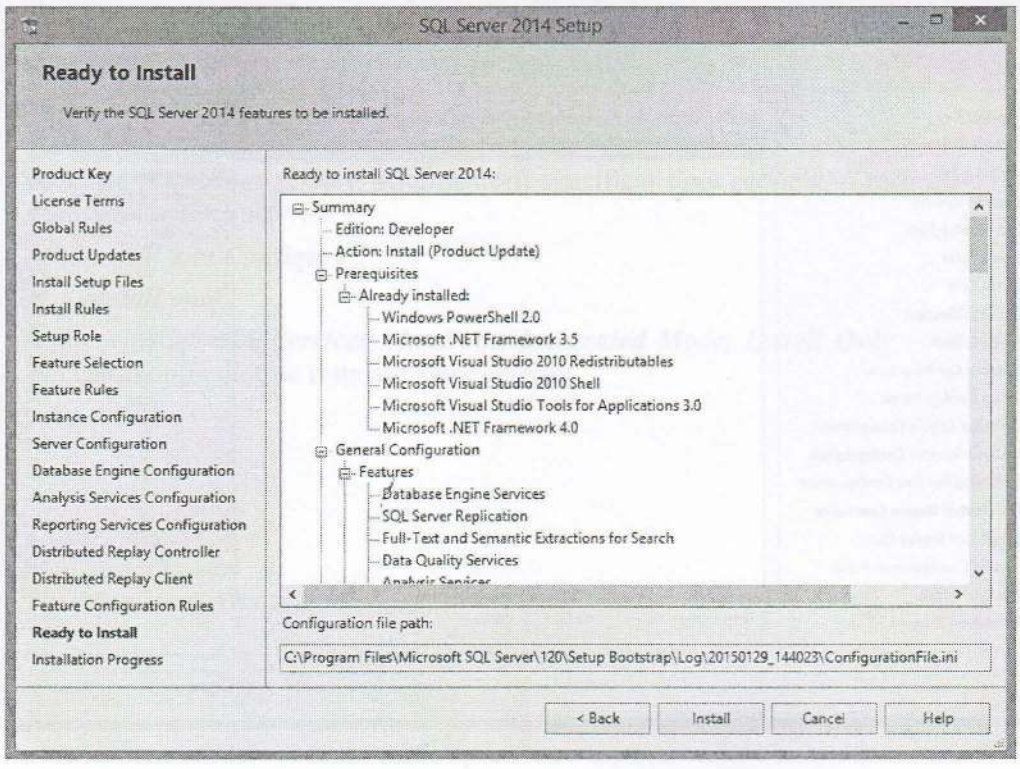
17. La următorul pas, ***Distributed Relay Controller,*** se specifică cel puțin un utilizator cu rolul de administrator Analysis Server la nivel de sistem de operare. După setările respective, se apasă butonul **Next**.



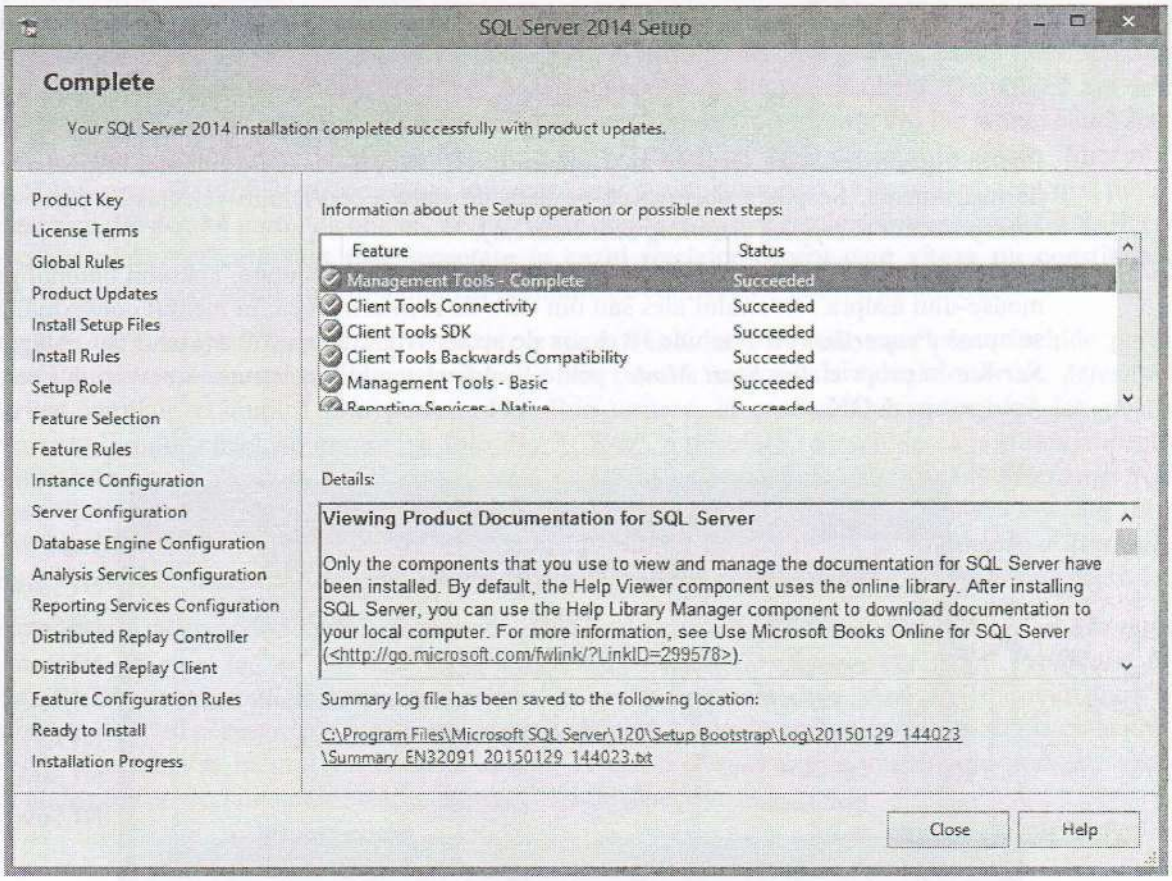
18 La pasul ***Distributed Relay Client*** se lasă proprietățile implicite. Pentru a continua se apasă butonul **Next**



19. Următoarea fereastră, ***Ready to install,*** apare în cazul în care sistemul este pregătit de instalare. Utilizatorul este informat despre componentele selectate pentru instalare și configurația lor pentru a avea posibilitatea de a verifica încă o dată înainte de a fi lansat procesul de instalare propriu-zis. După apăsarea butonului **Install,** procesul de instalare este lansat.



20. Se deschide fereastra ***Installation Progress,*** unde poate fi urmărită parcurgerea procesului de instalare. Este afişată o bară de progres, iar mai jos se scriu comentariile despre starea curentă a procesului de instalare.
21. După ce procesul de instalare se termină cu succes, apare pagina ***Complete*** cu mesajul: „***Your SQL Server 2014 installation completed successefully***", iar mai jos se afişează rezumatul stării de instalare pentru fiecare componentă a SGBD-ului. În partea de jos a ferestrei, se oferă un link spre fişierul *log* care conţine informaţii sumare despre procesul de instalare. Pentru a închide asistentul de instalare, se apasă butonul **Close.**



Uneori, pentru pornirea corectă a sistemului SQL Server, după instalare se recomandă restartarea calculatorului.

Procesul de instalare a versiunii SQL Server 2014 Express Edition se deosebeşte puţin de cel descris anterior

1.2.4. Iniţializarea serviciilor

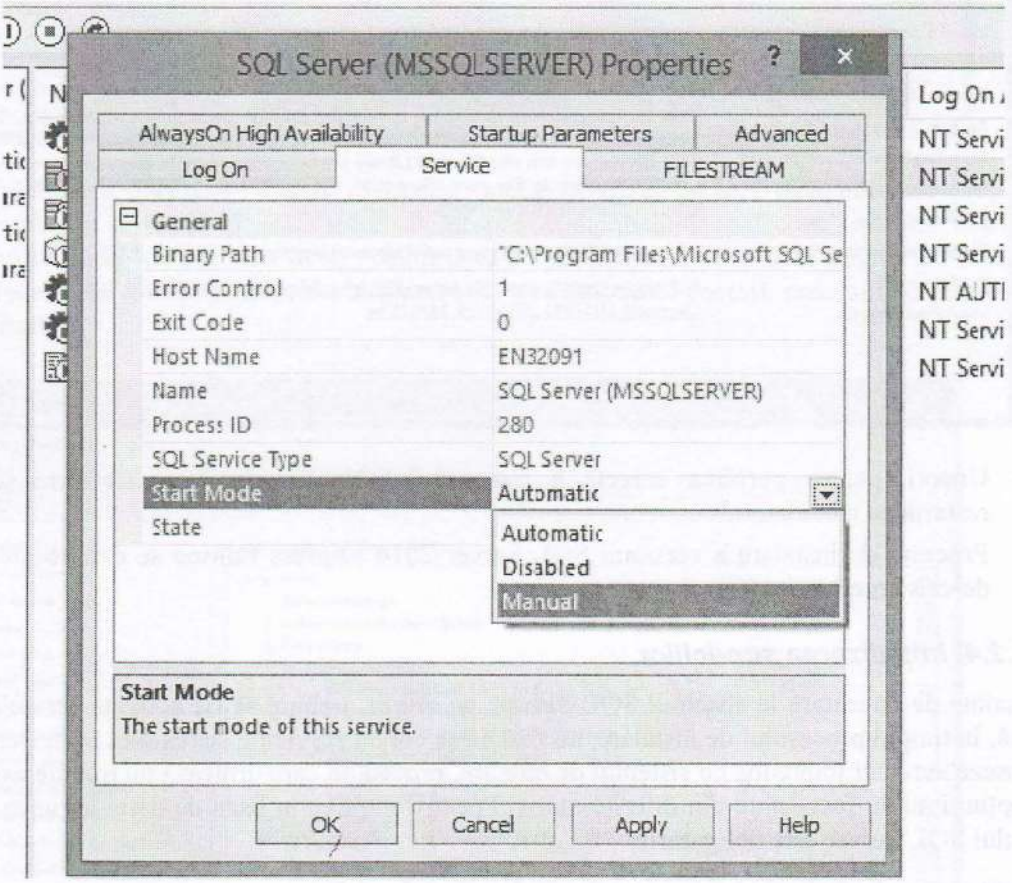
Înainte de conectare la sistemul SQL Server, pe server, trebuie să fie activate serviciile sistemului [4]. Dacă, în timpul procesului de instalare, au fost alese corect serviciile sistemului SQL Server, necesare să se lanseze automat împreună cu sistemul de operare, procedura care urmează nu trebuie executată. Dacă aceste opţiuni nu au fost setate din diferite motive, pentru intrarea în baza de date, iniţierea unor servicii SGBD-ului SQL Server este obligatorie.

Pentru aceasta, trebuie să fie executaţi următorii paşi:

1. Se activează ***Start Screen*** şi se selectează opţiunea ***Apps***.
2. În lista aplicaţiilor disponibile se alege categoria ***Microsoft SQL Server 2014***.
3. Se selectează elementul ***SQL Server 2014 Configuration Manager***
   În acest punct, pe ecranul calculatorului, va fi prezentată următoarea fereastră:

4. Pentru a gestiona vreun serviciu al sistemului SQL Server, se apasă butonul respectiv pe bara de instrumente. Se poate observa că pe bara de stare a serviciului selectat a apărut mesajul corespunzător (***Stopped, Running sau Paused.***).

   Aceleaşi operaţii de lansare şi stopare a serviciului pot fi executate, apăsând butonul drept al mouse-ului asupra serviciului ales sau din bara de meniuri. Dacă, în meniul contextual apărut, se apasă ***Properties,*** se deschide fereastra de setări avansate pentru serviciul dat. Alegând fila ***Service*** în proprietatea ***Start Mode***, poate fi indicat modul de lansare a serviciului respectiv. Apoi se apasă **OK**.



5. Dacă, în loc de iniţializare (click pe butonul **Start Service**), se execută un click pe butonul **Pause Service** asupra serviciului respectiv, noi operaţiuni ale utilizatorilor nu vor fi executate, însă tranzacţiile executate în acel moment nu vor fi paralizate. Iar în cazul în care se va da un click pe butonul **Stop Service**, toate tranzacţiile vor fi finalizate şi utilizatorii vor fi deconectaţi de la serviciul respectiv.

Trebuie menţionat că activităţile cu ***SQL Server Configuration Manager*** (lansarea şi stoparea serviciilor de sistem) pot fi realizate numai de utilizatorii cu drepturi de administrator de sistem de operare.

1.3. Conexiunea la servere

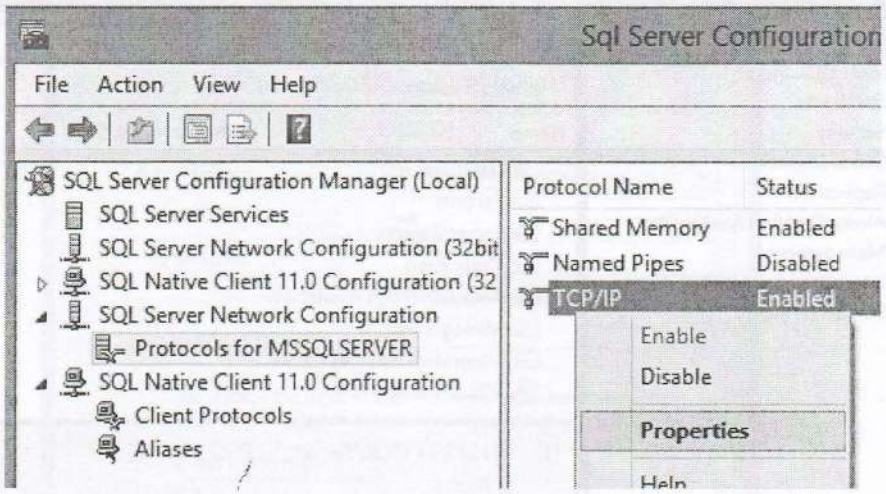
1.3.1. Înregistrarea serverelor

Pentru înregistrarea unui server îndepărtat, trebuie să se ţină cont de unele condiţii.

În comunicarea cu clienţii şi cu alte servere, MS SQL Server utilizează protocoale ***Shared Memory***, ***TCP/IP*** şi ***Named Pipes***. În funcţie de protocoalele de acces utilizate de SQL Server, există trei opţiuni de activare a conexiunii la distanţă: doar ***TCP/IP***, doar canalele dedicate, sau ambele. De obicei, are sens activarea doar a unui singur protocol, cel necesar. Activarea ambelor protocoale are loc atunci când, într-adevăr, e necesară conexiunea la server atât prin ***TCP/IP***, cât şi prin ***Canale Dedicate***. Microsoft recomandă folosirea ***TCP/IP*** deoarece acesta utilizează mai puţine porturi şi, prin urmare, sunt mai puţine canale deschise. Pe de altă parte, de obicei, ***TCP/IP*** este mai preferat pe reţele cu viteze mici ***LAN***, ***WAN***, în timp ce canalele dedicate sunt mai acceptate în cazul reţelelor pentru care viteza nu constituie o problemă, deoarece oferă multe funcţionalităţi şi opţiunile de configurare pot fi uşor folosite.

Implicit, ***Microsoft Windows*** activează programul ***Windows Firewall***, care poate închide portul 1433 pentru a preveni conectarea altor calculatoare, din reţeaua Internet, la calculatorul dat. ***Microsoft SQL Server,*** implicit, comunică prin portul 1433. Prin urmare, la configurarea sistemului dat, pentru urmărirea conexiunilor-client de intrare, ce folosesc ***TCP/IP***, e necesară redeschiderea portului respectiv sau modificarea portului de acces, altfel, conexiunile ***TCP/IP*** sunt imposibile. Atunci când ***Microsoft SQL Server*** este configurat pentru urmărirea conexiunilor client de intrare folosind canalele dedicate prin sesiuni tip **NetBios**. De exemplu SQL Server comunică, prin porturi TCP 139 şi 445 pentru ***Filestream.*** Prin urmare, conexiunile prin canalele dedicate sunt posibile atunci când sunt deschise porturi respective.

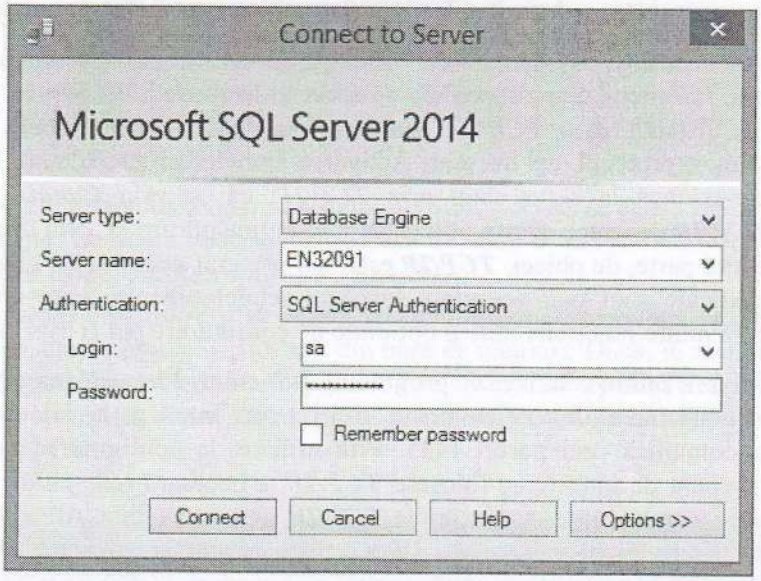
Proprietăţile de conexiune pot fi gestionate, dacă se deschide ***SQL Server Configuration Manager*** şi, în câmpul de obiecte, se setează ***SQL Server Network Configuration,*** apoi ***Protocols for MSSQLSERVER.*** În câmpul de lucru, apare lista de protocoale accesibile. Apăsând butonul drept al mouse-ului pe obiectul necesar, pot fi activate/dezactivate protocoale de comunicare, apelând la comenzile **Enable** sau **Disable** din meniul de context apărut. Alegând ***Properties*** in meniul de context, apare fereastra unde pot fi redactate unele setări ale protocolului respectiv.



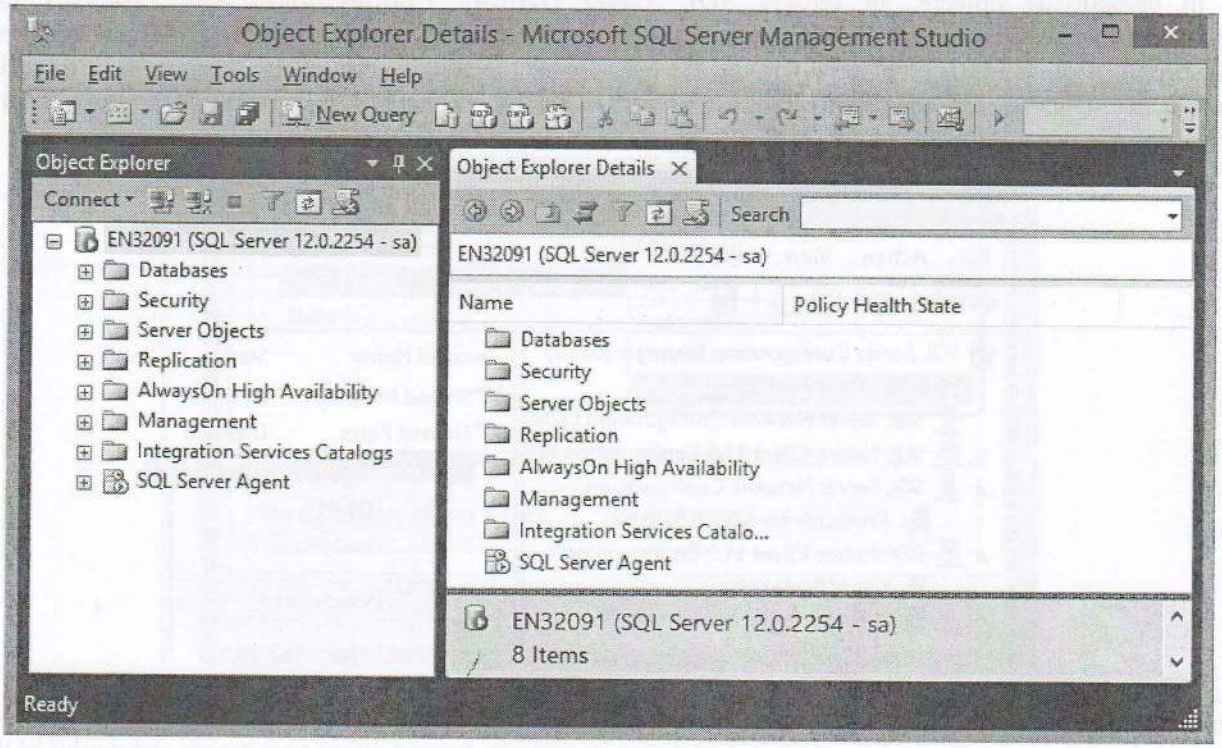
Atunci când are loc instalarea sistemului SQL Server, serverul local se înregistrează în mod automat. Funcţia de înregistrare a serverelor se aplică în cazul în care se doreşte accesarea sistemului SQL Server instalat pe alt calculator. Această procedură poate fi realizată utilizând ***SQL Server Management Studio***:

1. Se activează ***Start Screen*** şi se selectează opţiunea ***Apps***.
2. Se accesează opţiunea ***Programs***.

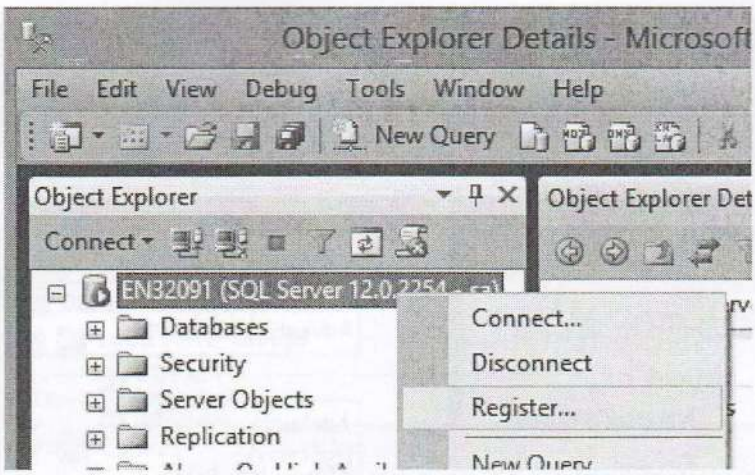
3. În lista aplicațiilor disponibile se alege categoria ***Microsoft SQL Server 2014***.
4. Apoi se trece și se selectează, cu un click, opțiunea ***SQL Server 2014 Management Studio***
5. După aplicarea unui click pe opțiunea ***SQL Server 2014 Management Studio***, se deschide o cutie de dialog de conectare a clientului (interfața grafică) la SQL Server. Aici se alege tipul serviciului la care se dorește conectarea, numele serverului, tipul de autentificare, loginul și parola. Ecranul lui este similar celui prezentat în figura de mai jos:



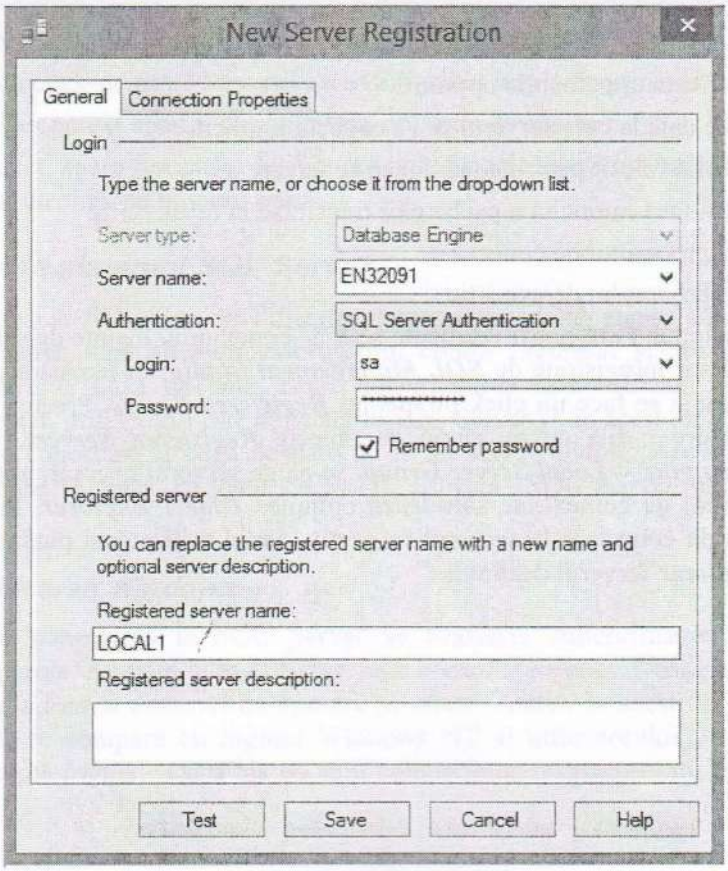
6. După apăsarea butonului **Connect,** apare fereastra de mai jos. Pentru a vizualiza partea dreaptă a ferestrei (***Object Explorer Deatails***), se apasă **F7**.



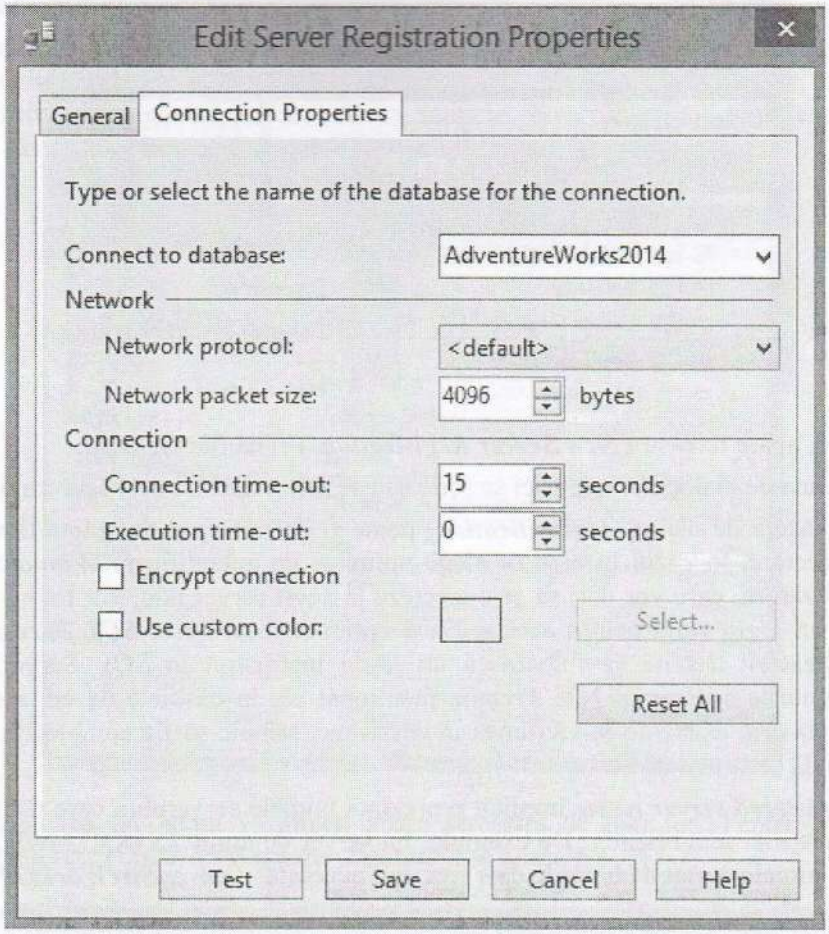
7. În fereastra respectivă, se face un click-drept pe server în ***Objest Explorer*** și din meniul de context se alege opțiunea ***Register***.



8. Pe ecran, apare fereastra ***New Server Registration***. Prima fila ***General.***
   - Căsuța de dialog ***Server.*** Aici se scrie sau se selectează numele serverului.
   - În căsuța de dialog ***Authentication,*** poate fi definit tipul de autentificare utilizat pentru conectare. În cazul în care se alege opțiunea de autentificare ***Windows authentication***, utilizatorii, care vor dori să se conecteze la acest server nou, vor trebui să intre în sistem cu un *login* valid pentru acesta. Dacă opțiunea aleasă este ***SQL Server Autentification***, utilizatorii trebuie să folosească un *login* înregistrat în SQL Server, independent de loginurile sistemului NT. Trebuie menționat că, în căsuțele de editare ***Login Name*** şi ***Password***, existente în secțiunea în chestiune, trebuie să fie completate *loginul* şi *parola* validă pentru acest server.
   - ***Registered server name*** implicit reprezintă numele serverului, care poate fi rescris cu un nume mai semnificativ. De exemplu, un server denumit *TSAKK32440* poate fi înregistrat cu numele purtat de baza de date frecvent accesată – ***AdventureWorks2014***.



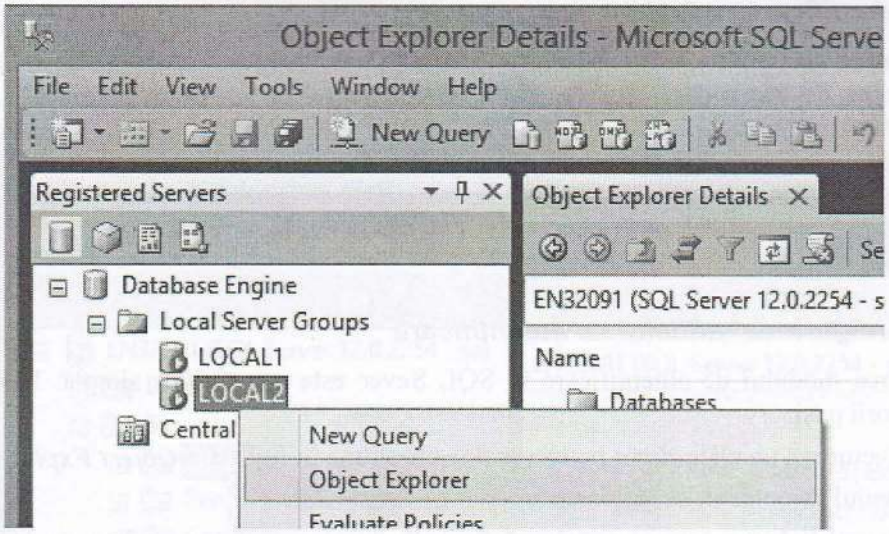
9. În fila alăturată, ***Connection Properties,*** se fixează proprietăţile de conexiune ale serverului.



Pot fi specificate următoarele opţiuni:

- Baza de date la care serverul se va conecta implicit.
- Protocolul folosit pentru conexiunea la server.
- Dimensiunea implicită a pachetelor transmise în reţea.
- Durata procesului de conectare.
- Durata procesului de execuţie.

10. Apăsând butonul **Test,** pot fi verificate setările conexiunii, înainte de salvare. Pentru a viziona lista serverelor înregistrate de ***SQL Management Studio***, în fereastra principală, se lansează meniul ***View*** şi se face un click pe meniul ***Registered Severs***. Precum se vede din figura de mai jos, apare o filă cu un câmp de obiecte ***Registered Servers***. Expandând elementul ***Database Engine ->Local Server Group,*** se alege serverul necesar, apoi se dă un click-drept şi, din meniul de context, se selectează opţiunea ***Object Explorer***. În consecinţă, apare un dialog-box de conectare la serverul respectiv, astfel utilizatorul putând gestiona în interfaţa ***Object Explorer*** serverul dezirabil.



Trebuie menţionat că, în funcţie de componentele instalate ale sistemului Microsoft SQL Server 2014, pot fi setate următoarele tipuri de servicii: ***Database Engine, Analysis Services, Reporting Services*** şi ***Integration Services***. Instanţele locale sunt setate automat la prima lansare a sistemului ***Management Studio***, după instalarea ***SQL Server***.

1.3.2. Excluderea unui server

Procesul de excludere a unui server este foarte simplu. Pentru aceasta, este suficientă executarea unui click cu butonul drept al mouse-ului pe imaginea serverului respectiv în fila ***Registered servers*** din partea dreaptă a ferestrei ***SQL Server Management Studio.*** În meniul de context apărut, se acţionează opţiunea ***Delete*** şi apoi se confirmă această alegere.

1.3.3. Modificarea înregistrării unui server

Pentru executarea acestei sarcini, trebuie parcursă următoarea cale:

1. Se aplică un click-drept pe serverul care va fi editat în fila ***Registered servers.***
2. În meniul de context apărut, se acţionează opţiunea ***Properties***.
3. În acest punct, apare fereastra, deja, cunoscută. Se selectează opţiunile necesare şi se confirmă alegerea cu un click pe butonul **Save.**

1.4. Loginuri în sistemul SQL Server

Pentru conectarea la SQL Server, utilizatorul trece prin două etape: autentificarea şi validarea permisiunii. Autentificarea identifică utilizatorul prin contul *loginului* şi are funcţia de verificare a condiţiilor de conectare la baza de date. Dacă autentificarea este realizată, adică, dacă *loginul* de autentificare este acceptat, utilizatorul este conectat la SQL Server şi, în acest punct, utilizatorul trebuie să obţină permisiunea de conectare la baza de date de la un server anume. Acest proces este realizat cu ajutorul conturilor *login* în baza de date ce va fi utilizată. SQL Server utilizează două tipuri de autentificare [4].

1.4.1. Autentificarea Windows

Atunci când la conectarea la SQL Server se foloseşte Autentificarea Windows, nu trebuie specificate *loginul* şi parola. Accesul la SQL Server este controlat de contul sau grupul Windows NT, care este autentificat la *loginarea* la sistemul de operare pe client. Astfel, în acest tip de autentificare, contul utilizatorului din SQL se compară cu *loginul* Windows NT al utilizatorului. În cazul în care ele sunt compatibile, SQL acceptă *loginul*. Dacă ele nu sunt compatibile, utilizatorul nu este conectat la sistemul SQL Server.

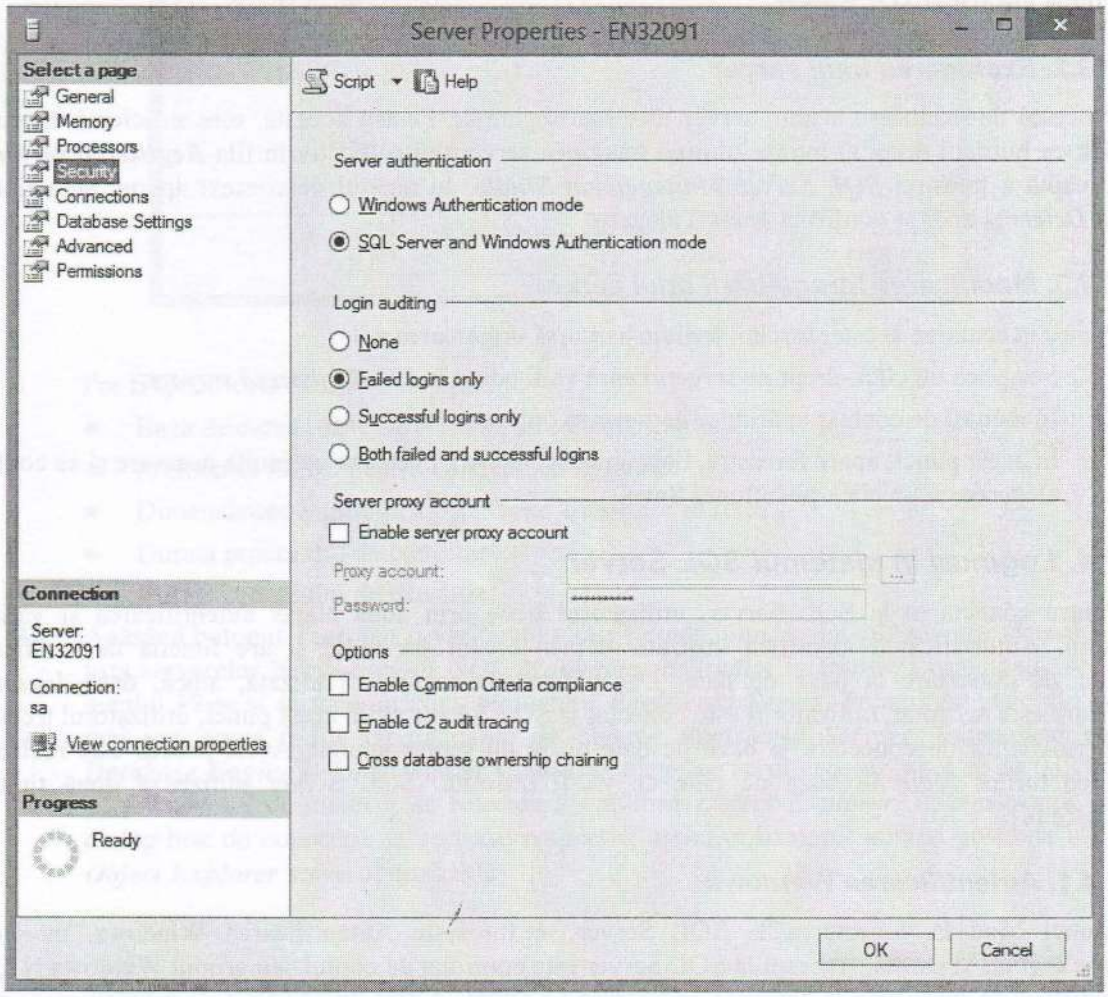
1.4.2. Autentificarea mixtă

Autentificarea mixtă este un serviciu de autentificare independent de sistemul de *loginare* al Windows, gestionat de însuși SQL Server. Dacă, la conectare, un utilizator furnizează un *login* SQL Server, acesta este autentificat, folosind autentificarea SQL Server. Însa trebuie să se țină cont de faptul că pentru a inițializa o conexiune la SQL Server, utilizatorul are nevoie să intre în mediul Windows, fapt ce implică, mai întâi verificarea accesului la nivel de sistem de operare. Pentru a gestiona serviciile SQL Server, pornirea și oprirea lor, utilizatorul SGBD-ului SQL Server trebuie să aibă permisiuni de administrator local al sistemului de operare.

1.4.3. Configurarea modului de autentificare

Configurarea modului de autentificare în SQL Sever este o activitate simplă. Pentru aceasta, se realizează următorii pași:

1. Se efectuează un click-drept pe serverul în chestiune în fereastra ***Object Explorer***.
2. În meniul de context, se acționează opțiunea ***Properties***.
3. În câmpul ***Select a page*** se selectează ***Security***. În acest punct, va apărea următoarea fereastră:
4. În fereastră, se marchează ***Windows Authentification mode*** sau ***SQL Server and Windows Authentification mode***, în funcție de tipul de autentificare care trebuie să-l asigure sistemul SQL Server.

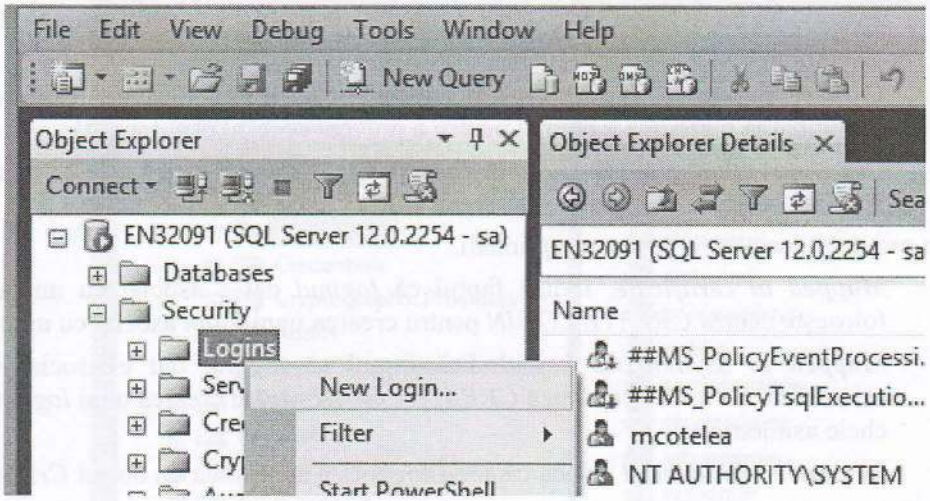


1.4.4. Crearea loginului propriu

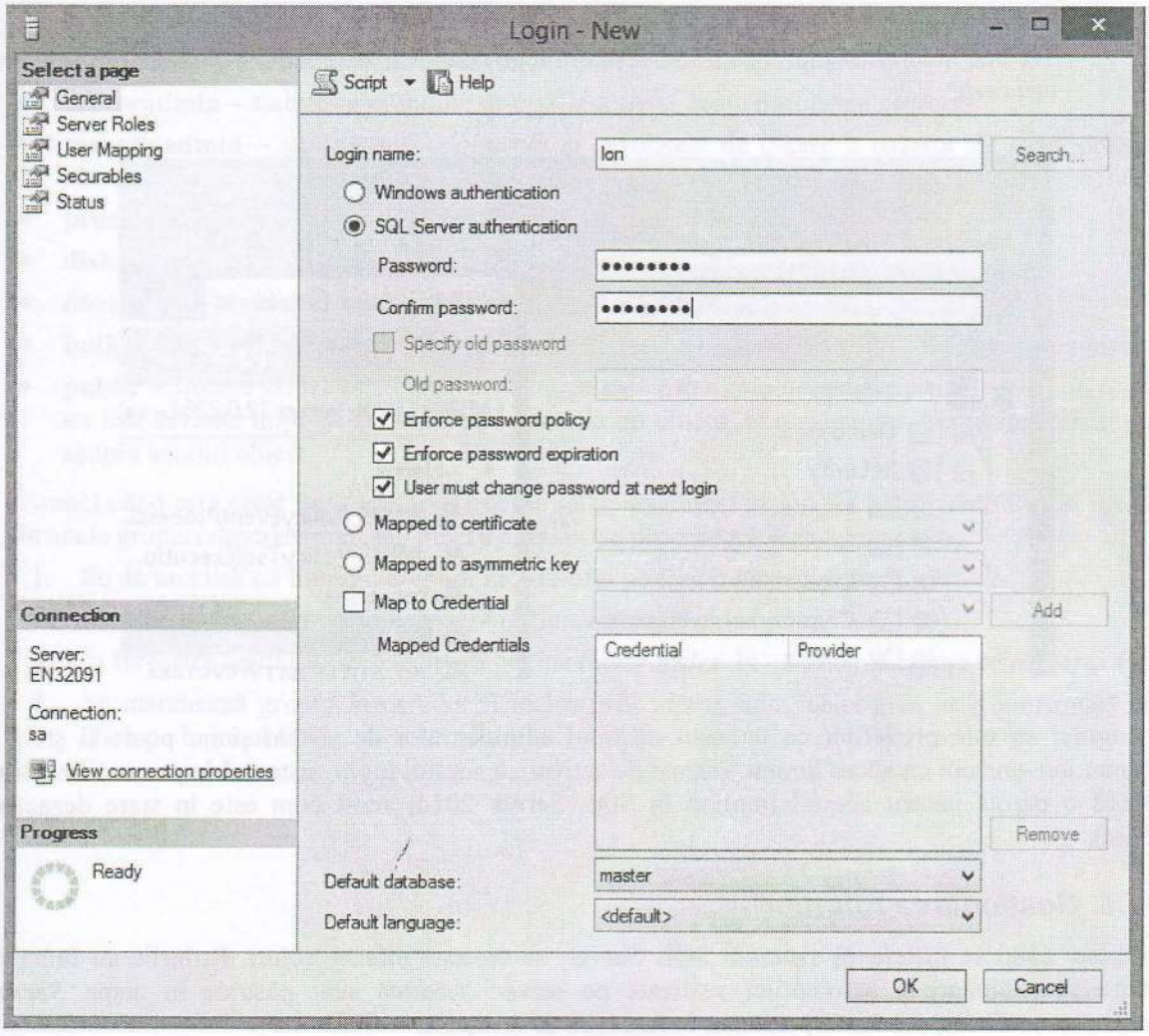
Dacă SQL Server utilizează modul de autentificare mixt, vor trebui create conturi care conțin nume și parole specifice pentru utilizatori. Pentru crearea conturilor, se parcurg următorii pași:

1. Se dă pe arbore un click pe semnul "+" din partea serverului care va recepționa un *login* nou.

2. Se aplică un click pe semnul "+" al mapei ***Security***.
3. Se face un click-drept pe elementul ***Logins***. În consecință, apare imaginea următoare:



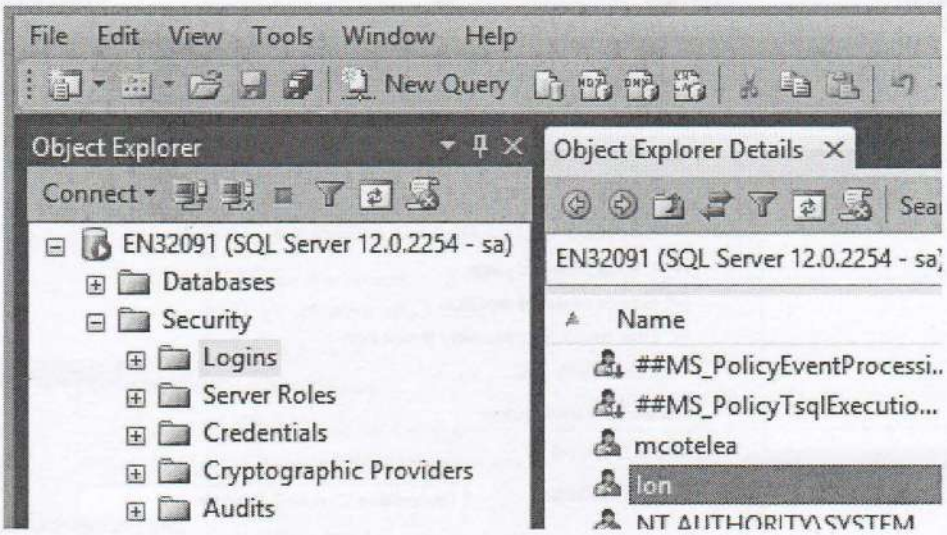
4. În meniul de context, se acționează opțiunea ***New Login***. Apare următoarea fereastră:



5. Se selectează opțiunea SQL Server Autentification.
6. În căsuța de editare ***Login Name,*** se introduce un nume al utilizatorului.

7. În căsuța de editare ***Password,*** se introduce o parolă a utilizatorului.
8. Apoi se setează următoarele radiobutoane:
   - ***Enforce password policy*** Se cer respectate normele de creare a parolei pentru utilizatorul dat. Aceasta este setarea implicită la alegerea autentificării ***SQL Server.*** Se recomandă să fie lăsată activă.
   - ***Enforce password expiration*** Activarea politicii timpului de expirare a parolei (schimbarea periodică a parolei pentru a preveni accesul neautorizat).
   - ***User must change password at next login*** SQL Server va cere setarea unei noi parole la prima accesare a contului nou-creat.
9. În cazul dat, următorii paşi sunt opționali.
   - ***Mapped to certificate.*** Indică faptul că *loginul* dat e asociat cu un certificat. Se foloseşte clauza *CREATE LOGIN* pentru crearea unui *login* asociat cu un certificat.
   - ***Mapped to asymmetric key.*** Indică faptul că *loginul* dat e asociat cu o cheie asimetrică. Se foloseşte clauza *CREATE LOGIN* pentru crearea unui *login* asociat cu o cheie asimetrică.
   - ***Mapped to Credential*** indică că acest *login* este asociat cu un obiect *Credential.*
10. Acest pas este opțional. În căsuța de selecție ***Default Database,*** se alege baza de date implicită la care utilizatorul va fi conectat după accesarea sistemului SQL Server şi inserarea unui *login* valid. În căsuța de selecție ***Default Language,*** se poate alege limba în care, pe ecran, vor apărea mesajele pentru utilizator.
11. După finalizarea selectărilor, se acționează butonul **OK**

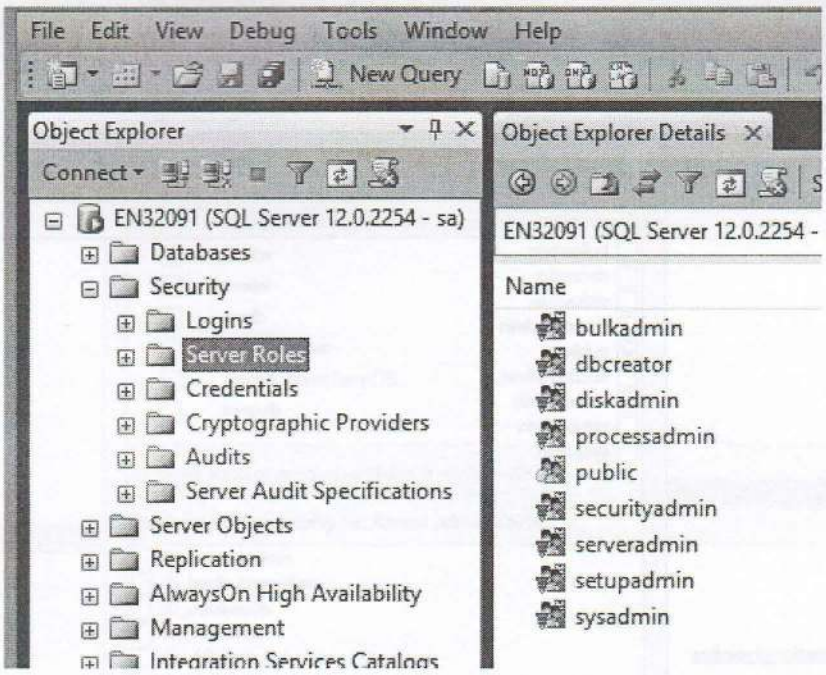
    Trebuie menționat faptul că a fost creat un *login* numit ***Ion***, precum e prezentat în figura ce urmează:



Loginul ***sa*** este predefinit ca un cont de tipul administrator de sistem şi nu poate fi şters sau transformat într-un cont cu acces limitat. Înainte de activare a acestui *login,* sistemul impune utilizatorul să stabilească o parolă pentru acesta. Implicit în SQL Server 2014, acest cont este în stare dezactivată (*Disabled*).

1.5. Gestionarea rolurilor

Atunci când se instalează sistemul SQL Server, se creează diferite roluri. Rolurile au funcția de prezentare şi gestionare a activităților realizate pe server. Acestea sunt păstrate în mapa ***Security***, elementul ***Server Roles***, precum e prezentat în fereastra ce urmează:

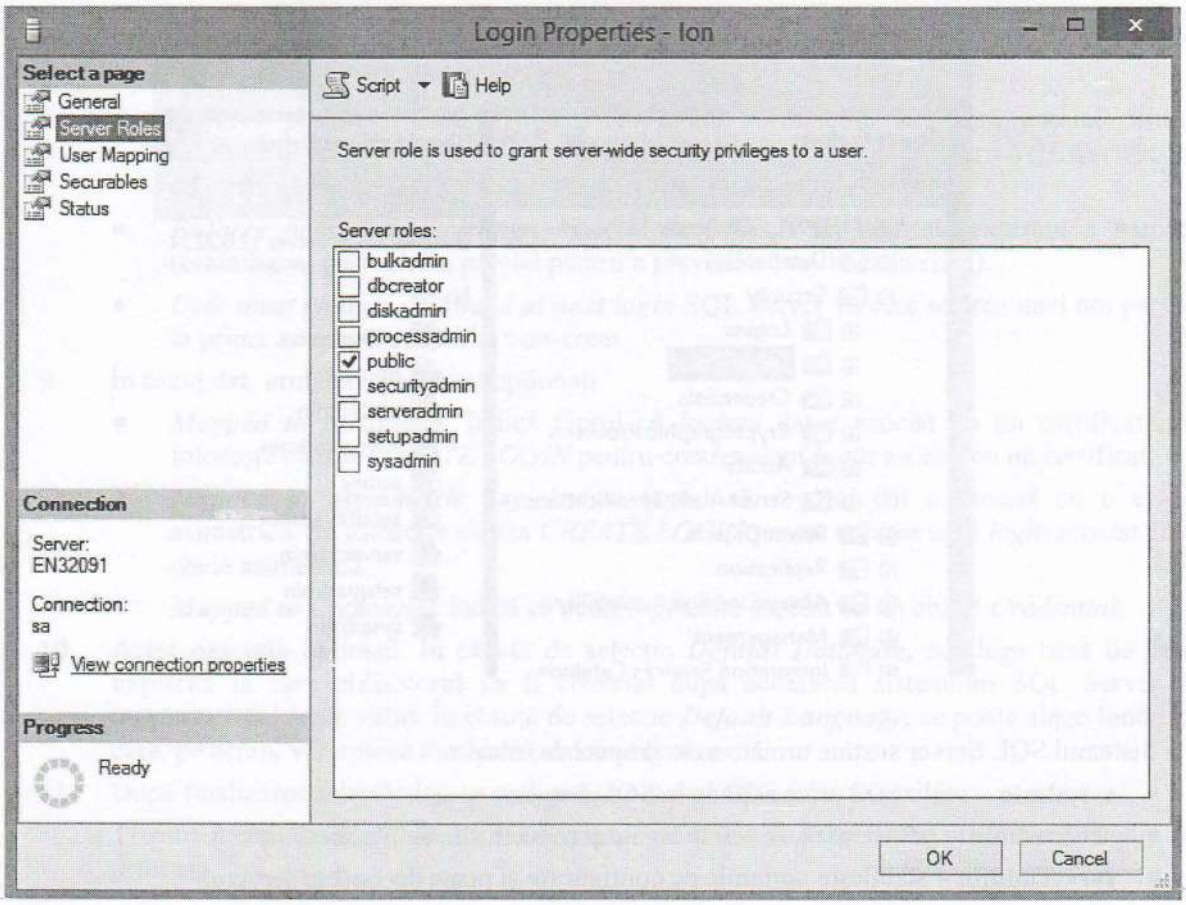


Sistemul SQL Server susține următoarele grupuri de roluri:

- **sysadmin** – realizează orice acțiune în SQL Server.
- **setupadmin** – gestionează serverele legate şi procedurile de inițiere.
- **serveradmin** – stabileşte opțiunile de configurare şi poate deconecta serverul.
- **securityadmin** – gestionează *loginurile* şi permisele de creare a bazelor de date, precum şi citirea fişierelor *log* a erorilor.
- **processadmin** –gestionează procesele care se execută în SQL Server.
- **diskadmin** – administrează fişierele de pe disc.
- **dbcreator** – creează şi modifică baza de date.
- **bulkadmin** – rolul serverului fixat poate lansa instrucțiunea *BULK INSERT* (inserarea masivă).
- **public** – fiecare utilizator al unei baze de date aparține rolului **public.** Dacă pentru un *login* nu au fost definite implicit drepturile de acces la un obiect, el moşteneşte drepturile rolului *public* asupra acestui obiect.

Atunci când este creat un *login* nou, trebuie să fie examinat faptul dacă utilizatorul va fi inserat în unul din acele grupuri descrise mai sus. Pentru aceasta, se procedează în felul următor:

1. Se dă un click cu butonul drept al mouse-ului pe *loginul* înregistrat
2. În meniul de context, se acționează opțiunea ***Properties***.
3. În fereastra deschisă, se selectează pagina ***Sever Roles***. În consecință, apare următoarea figură:
4. Se marchează grupul în care va fi inclus utilizatorul, apoi, selectarea este confirmată cu un click pe butonul OK.



În afară de determinarea grupului din care va face parte utilizatorul nou, există şi sarcina de stabilire a permiselor de acces la baza de date. Adică, se specifică ce baze de date, deja, create le va putea accesa utilizatorul nou şi ce permisiuni de acces va avea acest utilizator.

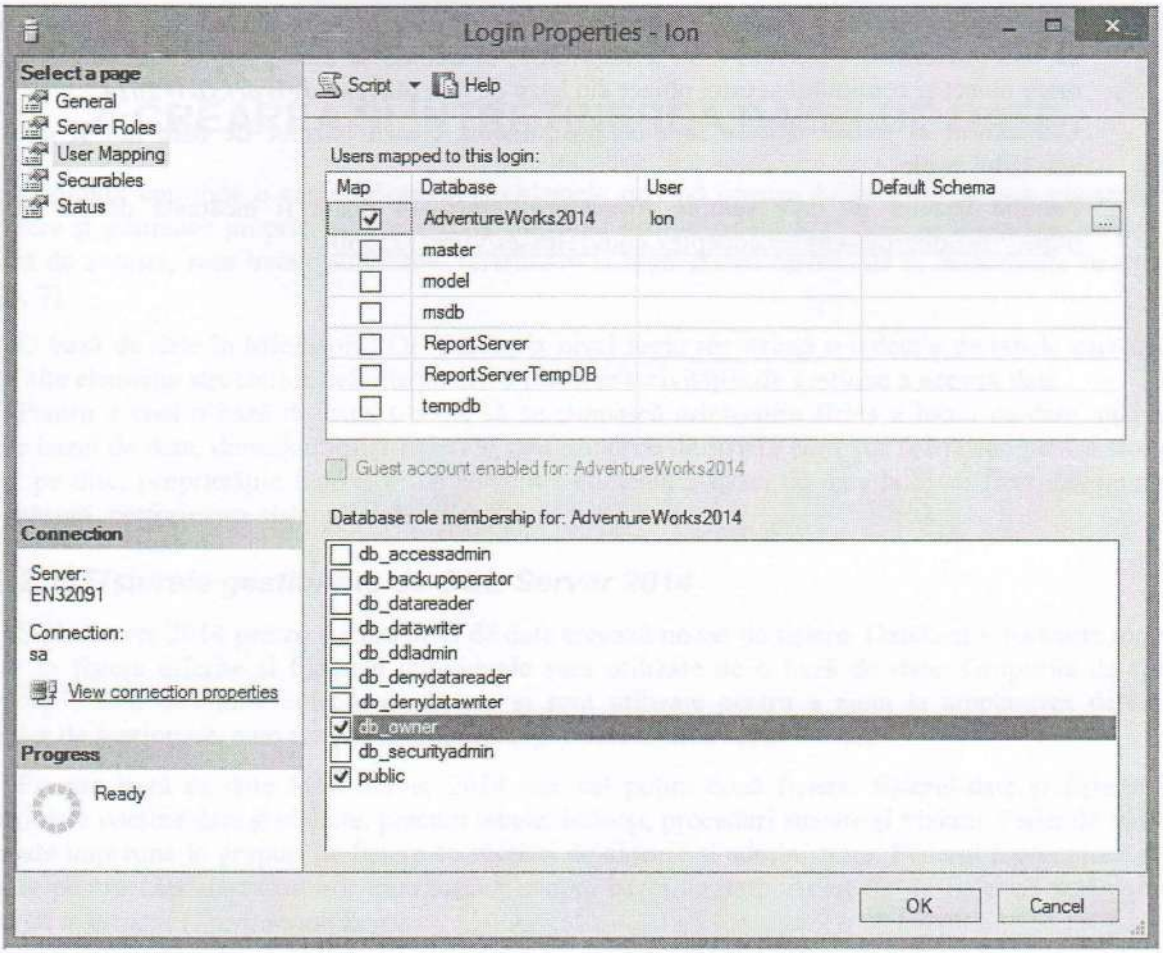
In afara de sistemul de gestiune a permisiunilor la nivel de server, ***Server Roles***, există sistemul de definire a accesului la nivel de bază de date. Un *login* poate fi declarat cu o bază de date sau mai multe baze de date ca un utilizator (*User*) de bază de date. Fiecărui utilizator din baza de date i se aloca un anumit Database Role.

Această sarcină, de asemenea, este uşor realizabilă:

1. Se face un click cu butonul drept al mouse-ului pe *loginul* recent creat.
2. În meniul de context, se acţionează opţiunea ***Properties***.
3. În fereastra apărută, se selectează pagina ***User Mapping***. Ca urmare, se obţine fereastra de mai jos:

   Tabelul de mai sus conţine următoarele câmpuri:

   - **Map** – setarea permisiunii de acces la baza de date respectivă pentru *loginul* dat.
   - **Database** – enumerarea tuturor bazelor de date disponibile pe server.
   - **User** – se specifică asocierea utilizatorului bazei de date cu *loginul*. Implicit, utilizatorul bazei de date are acelaşi nume ca şi *loginul*.
   - **Default Schema** - se specifică schema implicită a utilizatorului. În cazul de faţă, au fost definite roluri şi scheme implicite la *loginul* ***Ion*** pentru baze de date ***Sample – AdventureWorks2014***.



4. Selectările necesare sunt confirmate cu un click pe butonul **OK**.

În exemplul prezentat mai sus, utilizatorul cu *loginul* ***Ion*** este definit la nivel de server, însă la nivel de bază de date sunt utilizate definiția schemei şi rolurile de acces la schemă. În exemplul cu baza de date ***AdventureWorks2014,*** există un set de tabele cu informație despre angajați. E comod să se aloce aceste tabele la o schemă (în cazul dat ***HumanResources***) şi să se deschidă accesul la această schemă (grup de tabele) numai pentru anumiți utilizatori. Crearea schemelor va fi analizată în secțiunile ulterioare.

Întrebări de control:

1. Avantajele utilizării tehnologiilor client-server în gestiunea bazelor de date.
2. Resursele minimale hardware necesare pentru lucrul cu SQL Server 2014.
3. Componentele logice ale sistemului SQL Server.
4. Formele de autentificare în SQL Server 2014.
5. Roluri de utilizatori de server în SQL Server 2014, utilitatea, importanța.
6. Scheme în SQL Server 2014, utilitatea, importanța.

Sarcini practice:

1. Să se instaleze SQL Server 2014, urmărind procedurile descrise în lucrarea dată.
2. Să se înregistreze serverul pe calculatorul disponibil, alegând tipul de autentificare SQL.
3. Să se creeze trei *loginuri* de utilizatori care să posede SQL Autentificarea.
   Să se aloce primului utilizator acces la schema ***HumanResources*** din baza de date ***AdventureWorks2014***. Utilizatorul dat trebuie să aibă posibilitatea să citească şi să editeze datele din tabelele schemei respective.

Utilizatorul al doilea trebuie să aibă acces numai la citirea datelor din schema ***Sales*** din baza de date ***AdventureWorks2014***. În acelaşi timp, el trebuie să aibă posibilitatea să gestioneze orice obiect şi conținutul acestui obiect din baza de date ***AdventureWorksDW2014***.

Utilizatorul al treilea trebuie să aibă posibilitatea creării bazelor de date noi şi definirii accesului la ele

Familia bazelor de date sample AdventureWorks2014 poate fi încărcată de pe adresa: https://msftdbprodsamples.codeplex.com/releases/view/125550

2. CREAREA ŞI ÎNTREŢINEREA BAZEI DE DATE

Capitolul cuprinde o caracterizare şi problemele privind crearea bazelor de date, a planurilor de întreţinere şi gestiunea proprietăţilor acestora, utilizând instrumentele ***SQL Server Management Studio***. În afară de aceasta, sunt tratate subiectele referitoare la organizarea rezonabilă şi mentenanţa bazelor de date [6, 7].

O bază de date în Microsoft SQL Server la nivel logic reprezintă o colecţie de tabele care conţin date şi alte elemente structurale definite pentru a menţine activităţile de gestiune a acestor date.

Pentru a crea o bază de date, trebuie să se cunoască arhitectura fizică a bazei de date, mai întâi numele bazei de date, dimensiunea şi fişierele sau grupurile de fişiere care vor fi folosite pentru stocarea datelor pe disc, proprietăţile fişierelor. Organizarea eficientă a bazei de date la nivel fizic determină, în mare măsură, performanţa sistemului de calcul.

2.1. Fişierele gestionate de SQL Server 2014

SQL Server 2014 pentru fiecare bază de date creează un set de fişiere. Datele şi informaţia *log* sunt stocate în fişiere diferite şi fişierele individuale sunt utilizate de o bază de date. Grupurile de fişiere (*filegroups*) sunt denumite colecţii de fişiere şi sunt utilizate pentru a ajuta la amplasarea datelor şi sarcinilor de gestionare, cum ar fi operaţiile *backup* şi restabilirea bazei de date.

Fiecare bază de date SQL Server 2014 are, cel puţin, două fişiere: fişierul-date şi fişierul *log*. Fişierul-date conţine date şi obiecte, precum tabele, indecşi, proceduri stocate şi viziuni. Fişierele-date pot fi grupate împreună în grupuri de fişiere cu scopuri de alocare şi administrare. Fişierul *log* conţine datele necesare pentru executarea tuturor tranzacţiilor asupra bazei de date. Acest tip de fişier se mai numeşte *Jurnal de tranzacţii* (*Transaction Log*).

Bazele de date SQL Server 2014 pot conţine trei tipuri de fişiere, precum urmează:

Primar

Fişierul primar de date conţine date pentru startarea bazei de date şi indică către alte fişiere în baza de date. Datele utilizatorului şi obiectele pot fi păstrate în acest fişier sau în fişierul de date secundar. Fiecare bază de date are un singur fişier-date primar. Pentru acest tip de fişiere se recomandă extensia ***.mdf***.

Secundar

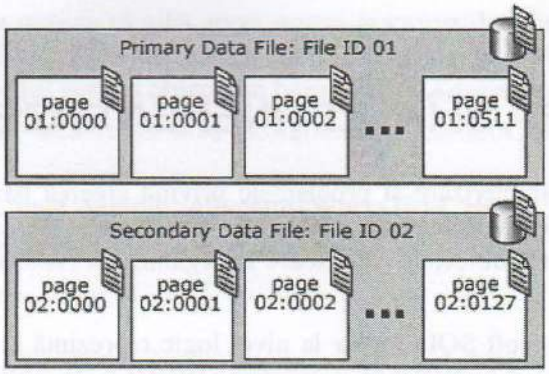
Fişierul-date secundar este opţional, este definit de utilizator şi păstrează date. Fişierele secundare pot fi utilizate pentru a răspândi datele pe mai multe discuri plasând fiecare fişier pe unitate diferită de disc. Adiţional, dacă baza de date excede mărimea maximală pentru un singur fişier Windows, poate fi utilizat fişierul-date secundar, astfel baza de date având posibilitatea să crească. Pentru acest tip de fişiere, se recomandă extensia ***.ndf***.

Log

Fişierul *log* conţine date folosite pentru restabilirea bazei de date. Trebuie să existe, cel puţin, câte un fişier log pentru fiecare bază de date. Pentru acest tip de fişiere, se recomandă extensia ***.ldf***

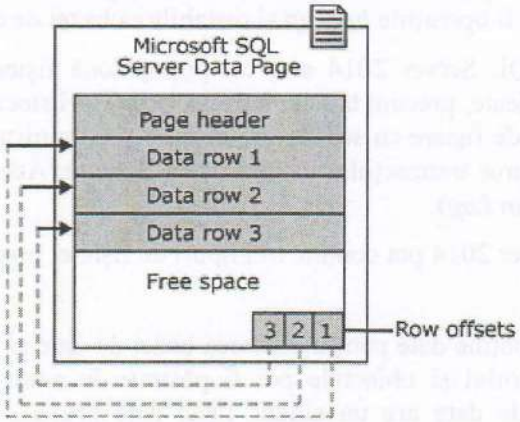
Principala unitate de păstrare a datelor în SQL Server 2014 este pagina. Spaţiul de disc alocat pentru fişierul-date (***.mdf*** sau ***.ndf***) în baza de date este divizat logic în pagini numerotate consecutiv de la 0 la *n*. Operaţiile de intrare/ieşire pe disc se realizează la nivel de pagini. Adică SQL Server citeşte şi scrie pagini întregi de date.

Prima pagină a fiecărui fişier este pagina-antet ce conţine date despre atributele acestuia. Următoarele câteva pagini, de asemenea, conţin date de sistem, ca, de exemplu, schemele de alocare. Una din paginile sistemului, stocată atât în fişierul primar, cât şi în primul fişier-log este pagina *boot* a bazei de date ce conţine date despre atributele acesteia.



În SQL Server, dimensiunea paginii este de 8 Kb, ceea ce semnifică faptul că o bază de date conţine 128 de pagini per 1 Mb. Fiecare pagină începe cu un antet de 96 octeţi, care e folosit pentru stocarea informaţiei de sistem despre pagina respectivă. Această informaţie include: numărul paginii, tipul ei, volumul spaţiului liber pe pagină şi ID-ul asociat obiectului, proprietar al acestei pagini.

Rândurile de date sunt amplasate pe pagină în mod secvenţial, începând imediat după antet. Tabelul de adrese ale rândurilor (*row offset table*) începe la sfârşitul fiecărei pagini, şi acest tabel conţine câte o intrare pentru fiecare rând de pe pagină. Fiecare celulă arată cât de departe, de începutul paginii, se găseşte primul octet al rândului. Intrările în tabelul de adrese sunt în ordinea inversă ordinii rândurilor pe pagină.



Extents reprezintă o consecutivitate de 8 pagini şi este utilizată pentru gestiunea eficientă a paginilor. Toate paginile sunt păstrate în *extenturi*. Cele 8 pagini ocupă 64 Kb, prin urmare baza de date SQL Server are câte 16 *extenturi* per 1Mb.

Pentru a optimizarea spaţiului alocat, SQL Server nu alocă *extenturi* întregi pentru tabelele cu un volum de date mic. SQL Server are 2 tipuri de *extenturi*:

- ***Extenturi uniforme*** – aparţin unui singur obiect, toate cele 8 pagini pot fi utilizate doar de obiectul proprietar.
- ***Extenturi mixte*** – pot fi partajate de până la 8 obiecte. Fiecare din aceste 8 pagini din *extent* pot avea proprietari diferiţi.

Gestiunea paginilor şi *extenturilor* depăşeşte limitele lucrării prezente.

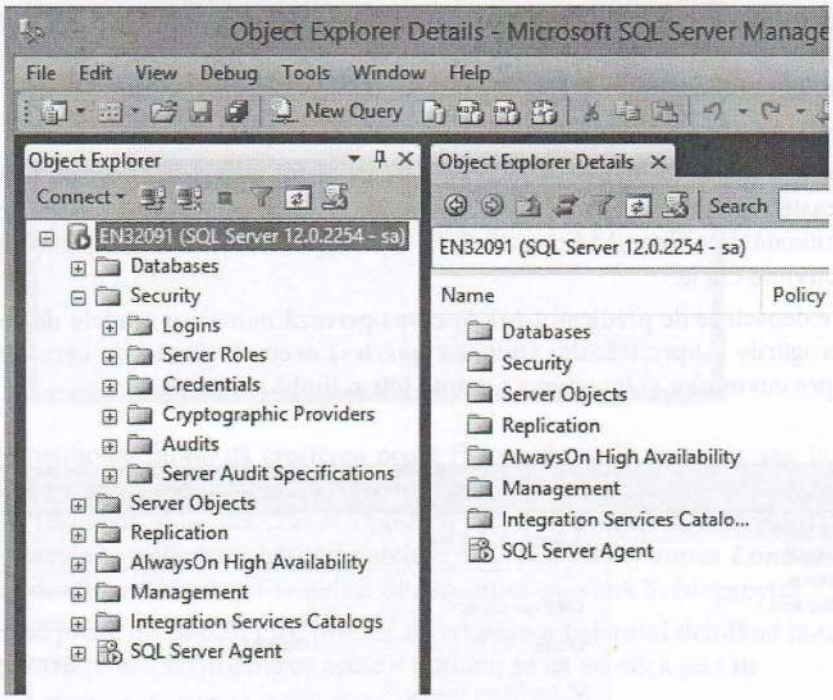
2.2. Crearea bazei de date

2.2.1. Lansarea SQL Server Management Studio pentru crearea bazei de date

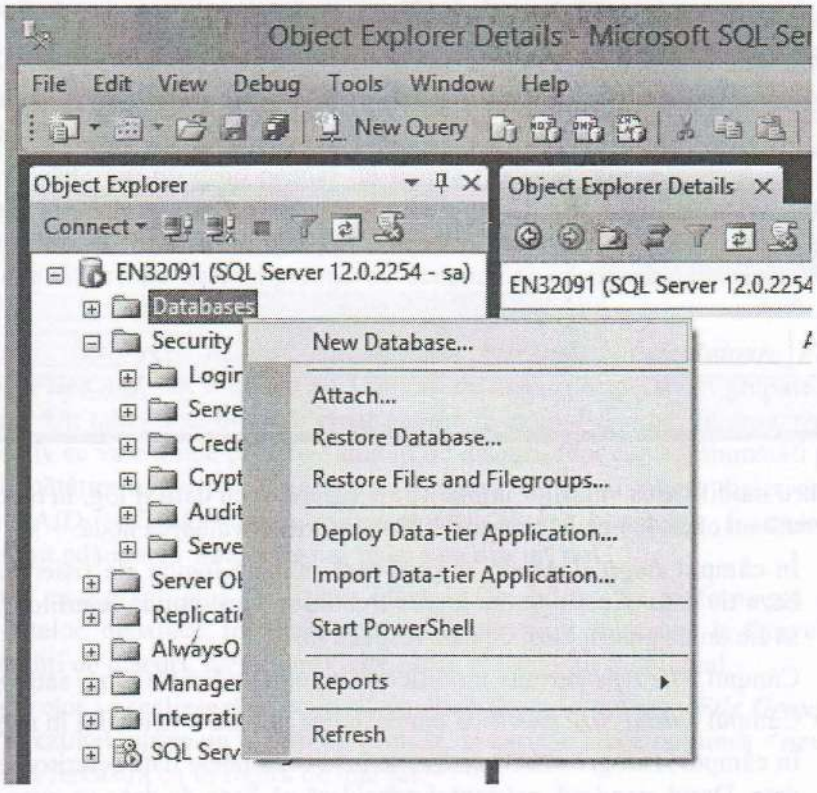
Crearea bazei de date se va realiza în ***SQL Server Management Studio***. Pentru aceasta, se procedează în modul următor:

1. Se activează ***Start Screen*** şi se selectează opţiunea ***Apps***.
2. Se accesează opţiunea ***Programs***.
3. În lista aplicaţiilor disponibile se alege categoria ***Microsoft SQL Server 2014***.

4. Apoi se trece şi se selectează, cu un click, opţiunea ***SQL Server 2014 Management Studio***
5. În fereastra ***Object Explorer***, serverul disponibil se extinde, aplicând un click pe semnul "+" din stânga lui. În rezultat, se expandează un arbore de obiecte ce alcătuiesc serverul respectiv.



6. În acest punct, se face un click cu butonul drept al mouse-ului asupra opţiunii ***Databases***. Pe ecranul calculatorului, va fi reprezentată următoarea fereastră:



7. În meniul de context, se efectuează un click pe opţiunea ***New Database***. Ca rezultat, pe ecranul calculatorului, va apărea fereastra de creare a bazei de date noi.

2.2.2. Afişarea şi modificarea proprietăţilor bazei de date (General)

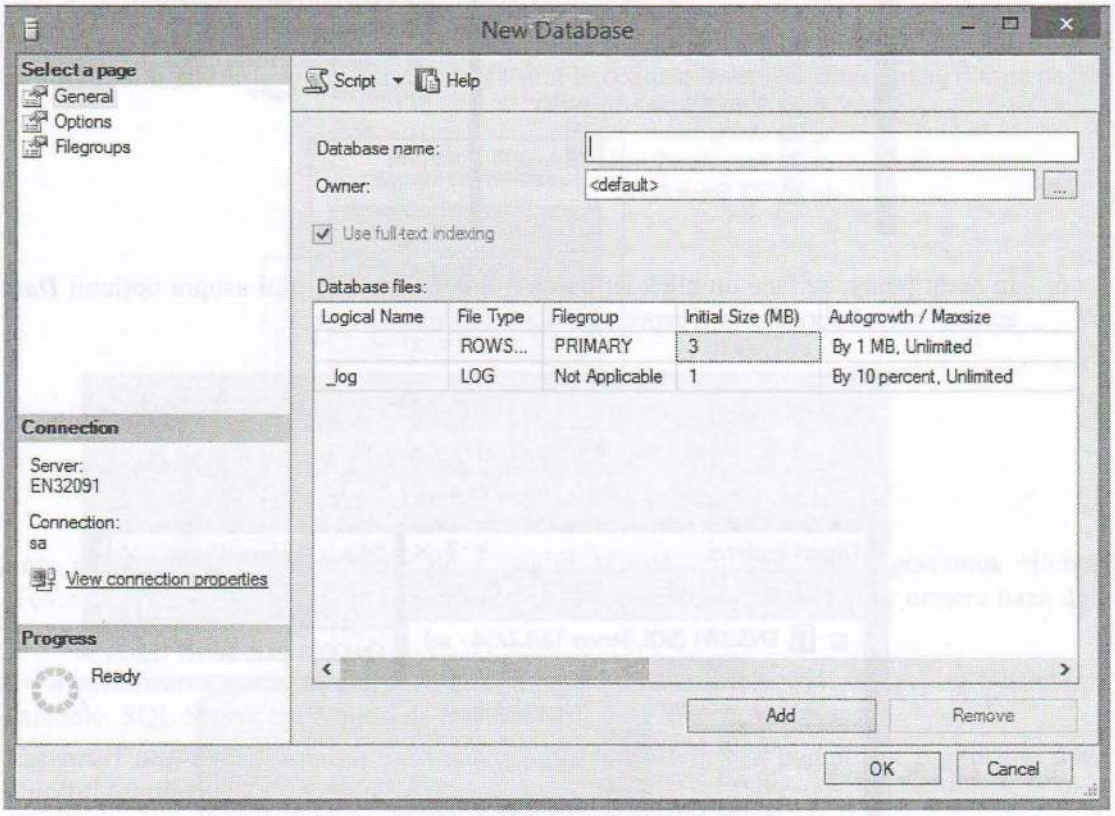
8. În căsuţa de editare ***Database name*** de pe pagina ***General***, se introduce numele bazei de date. Automat, sunt create fişierul de date primar şi fişierul *Log* cu numele specificat în calitate de prefix. Aceste obiecte pot fi vizionate în tabelul ***Database Files***.

   Pentru crearea bazei de date, care acceptă toate valorile implicite, este suficient să se insereze numele ei şi să se facă un click pe butonul **OK**, în caz contrar, se continuă cu următoarele activităţi opţionale.

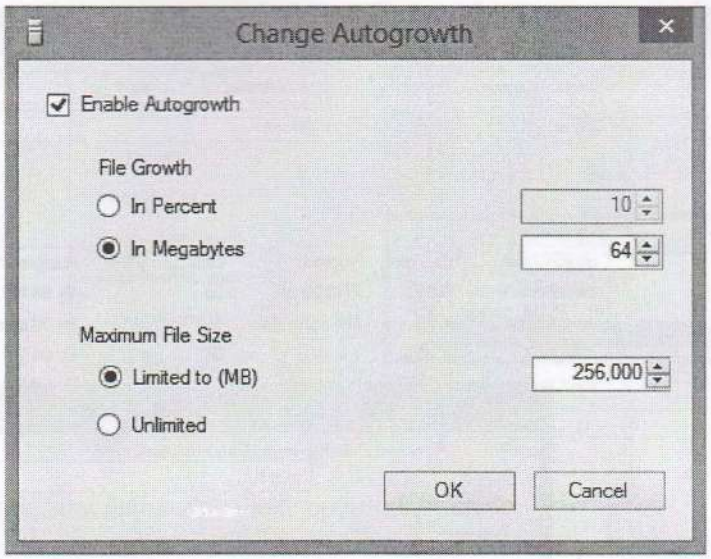
   Astfel, bazei create îi pot fi fixate proprietăţi speciale:

9. Pentru a activa căutarea în text în baza de date, se bifează caseta ***Use full-text indexing.*** Această opţiune permite indexarea rapidă şi flexibilă a textului păstrat în baza de date gestionată de sistemul Microsoft SQL Server, indexare necesară pentru interogările orientate pe cuvinte-cheie.

   Spre deosebire de predicatul *LIKE*, care operează numai cu modele de secvenţe de caractere, interogările asupra textului (*full-text queries*) execută căutări de caracter lingvistic, operând asupra cuvintelor şi frazelor construite într-o limbă concretă.



10. Pentru modificarea valorilor implicite ale fişierelor cu date şi log, în tabelul ***Database files*** se execută un click în celula respectivă şi se inserează valoarea nouă.
    - În câmpul ***Logical name,*** se inserează numele logice ale fişierelor fizice ce constituie baza de date. Aceste nume logice trebuie să corespundă regulilor SQL Server şi trebuie să fie unice pentru baza de date respectivă.
    - Câmpul ***File type*** permite modificarea tipului de fişiere (*data* sau *log*).
    - Câmpul ***Initial Size*** specifică dimensiunea iniţială a fişierului în megaocteţi.
    - În câmpul ***Autogrowth/Maxsize***, se inserează unele date referitoare la creşterea bazei de date. Drept standard, asistentul consideră că baza de date va avea o creştere de 10% în raport cu mărimea curentă sau 1MB.



Trebuie menţionat faptul că creşterea poate fi specificată în procente sau în megabytes, în funcţie de necesităţile proprii. Pentru aceasta, se apasă butonul la capătul drept al câmpului respectiv şi se marchează opţiunea respectivă în fereastra apărută. În cazul în care se doreşte specificarea mărimii maxime, este marcată opţiunea ***Limited to (MB)***, iar în căsuţa de editare de alături se indică dimensiunea maximă în Megabytes.

Tot aici, se poate deconecta proprietatea de creştere a fişierului debifând butonul ***Enable Autogrouwth.*** După configurarea acestor opţiuni, se dă un click pe **Ok.**

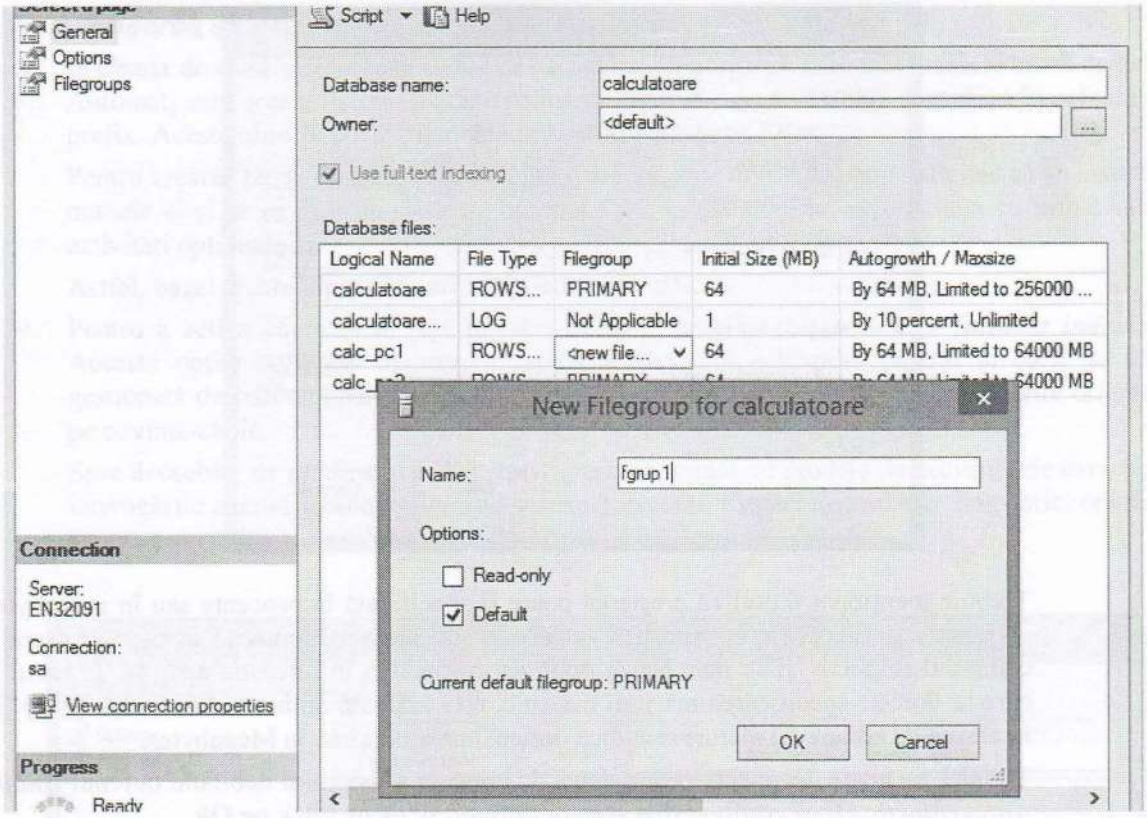
- În câmpul ***Path***, se indică alocarea fişierelor.
- În câmpul ***File Name***, se indică numele fizic al fişierului. Numele fizic trebuie să corespundă regulilor respective pentru denumirile fişierelor în mediul sistemului de operare dat. Dacă câmpul ***File Name*** nu este populat, sistemul, în mod automat, atribuie numele logic al fişierului în calitate de nume fizic, cu condiţia că acesta corespunde constrângerii sus-numite.
- Numărul de fişiere ale bazei de date poate fi modificat apăsând butoanele **Add** sau **Remove.**

Orice bază de date are un grup primar de fişiere (***primary filegroup***). Acest grup conţine fişierul primar de date şi unele fişiere secundare care nu au fost plasate în alte grupuri de fişiere. Grupurile de fişiere definite de utilizator (***user-defined filegroups***) pot fi create pentru a grupa fişiere de date împreună cu scopuri de gestionare, de alocare a datelor şi amplasare.

De exemplu, fie trei fişiere ***calculatoare_File2.ndf***, ***calculatoare_File3.ndf***, şi ***calculatoare_File4.ndf***, pot fi create pe 3 unităţi de discuri respectiv şi grupate în grupul de fişiere ***fgrup1***. Un tabel deci poate fi creat anume în grupul ***fgrup1***. Interogarea datelor din tabelul respectiv se va extinde pe cele 3 unităţi de discuri, fapt ce va îmbunătăţi performanţa. Aceeaşi îmbunătăţire a performanţei poate fi atinsă cu şi un singur fişier creat folosind tehnologiile RAID (*redundant array of independent disks*). În schimb, fişierele şi grupurile de fişiere permit adăugarea simplă de noi fişiere pe discuri noi.

Utilizarea mai multor grupuri de fişiere oferă o flexibilitate şi performanţă sporită şi la restabilirea datelor, deoarece, în paralel, pot rula operaţiuni de scriere în fişierele amplasate pe diferite unităţi de discuri, iar fiecare fişier poate fi restabilit individual.

Gruparea fişierelor se realizează executând un click-drept în coloana ***File Group*** a fişierului respectiv. Ca rezultat, apare un meniu de context, în care se alege opţiunea ***<new filegroup>*** şi se lansează o fereastră ca în figura de mai jos:



În cazul în care obiectele sunt create în baza de date, fără specificarea grupului de fişiere căruia aparţin, ele sunt atribuite grupului implicit. Într-un moment de timp, doar un grup de fişiere poate fi desemnat drept grup de fişiere implicit. Fişierele din grupul de fişiere implicit trebuie să fie destul de mari pentru menţinerea obiectelor noi neatribuite unui alt grup de fişiere. Grupul de fişiere *PRIMARY* este implicit.

Atunci când se defineşte un nou grup de fişiere implicit, fişierul primar va conţine numai tabele şi obiecte de sistem. Trebuie menţionat că fişierele *Transaction log* nu pot face parte nici dintr-un grup de fişiere.

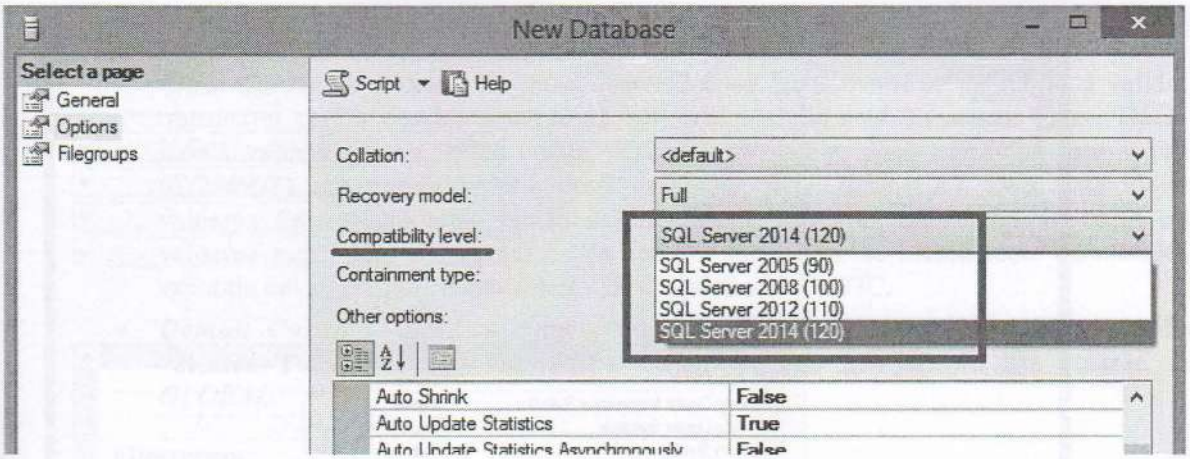
În cazul nostru, se specifică în calitate de grup de fişiere implicit - ***fgrup1***, bifând butonul respectiv.

2.2.3. Consultarea şi modificarea proprietăţilor bazei de date (Options)

Următoarea pagină ***Options*** determină caracteristicile bazei de date ce pot fi setate. Aceste opţiuni sunt unice pentru fiecare bază de date şi nu le afectează pe celelalte. Opţiunile sunt setate cu valori implicite la crearea unei baze de date noi.

Mai jos sunt prezentate opţiunile şi valorile lor implicite ce pot fi setate la crearea unei baze de date:

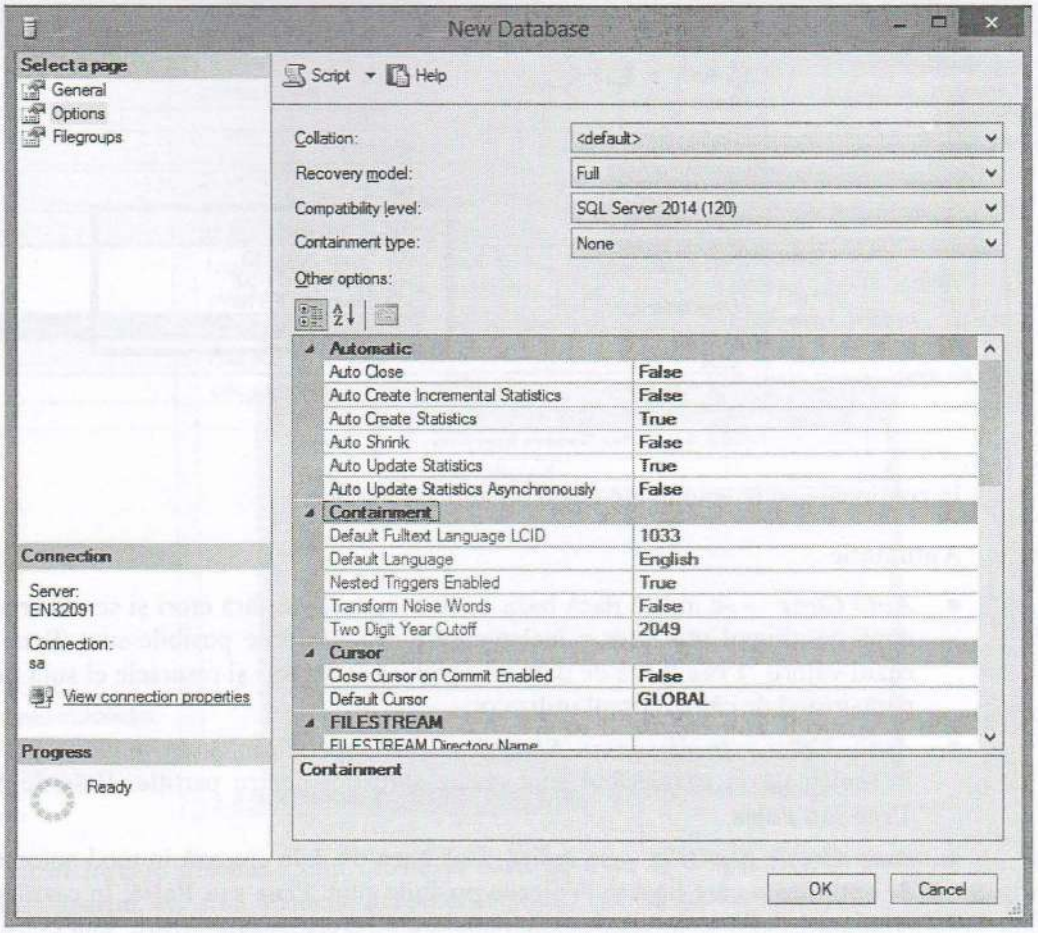
- ***Collation*** – specifică colaţionarea bazei de date, selectând valoarea respectivă din listă.
- ***Recovery model*** – specifică unul din următoarele modele de recuperare a bazei de date: ***Full, Bulk-Logged***, sau ***Simple.***
- ***Compatibility level*** – indică versiunea cea mai recentă a sistemului SQL Server ce susţine baza de date. Valori posibile: ***SQL Server 2014, SQL Server 2012, SQL Server 2008, SQL*** sau ***SQL Server 2005***.



În continuare, sunt enumerate unele opţiuni adiţionale:

Automatic

- ***Auto Close*** – se indică dacă baza de date se închide fără erori şi se eliberează resursele după ce ultimul utilizator a încheiat sesiunea. Valorile posibile sunt **True** şi **False.** În cazul valorii **True,** baza de date este închisă fără erori şi resursele ei sunt eliberate după părăsirea ei de către ultimul utilizator.
- ***Auto Create Incremental Statistics*** - se specifică dacă va fi utilizată opţiunea incrementală în cazul când sunt create statistici pentru partiţie. Valorile posibile sunt **True** sau **False.**
- ***Auto Create Statistics*** – se indică dacă baza de date creează în mod automat statisticile de optimizare care lipsesc. Valorile posibile sunt **True** sau **False**. În cazul valorii **True**, toate statisticile ce lipsesc, necesare pentru o cerere de optimizare, sunt construite în mod automat pe parcursul optimizării. Pentru mai multă informare, vezi *CREATE STATISTICS* (Transact-SQL).
- ***Auto Shrink*** – specifică dacă fişierele bazei de date sunt disponibile pentru compactizarea periodică. Posibile valori sunt **True** şi **False.** Pentru mai multă informare, vezi proprietatea ***AutoShrink.***
- ***Auto Update Statistics*** – se indică dacă baza de date automat modifică statisticile de optimizare depăşite. Valorile posibile sunt **True** şi **False.** În cazul valorii **True,** toate statisticile depăşite necesare pentru optimizarea interogărilor, sunt construite pe parcursul optimizării în mod automat. Pentru mai multă informare, vezi *CREATE STATISTICS* (Transact-SQL).



- ***Auto Update Statistics Asynchronously*** – în cazul în care este stabilită valoarea **True**, interogările ce iniţiază reînnoirea statisticilor învechite nu mai aşteaptă ca statisticile să fie reînnoite până la compilare. Dacă este fixată valoarea **False**, interogările ce iniţiază reînnoirea statisticilor învechite vor aştepta până la momentul când statisticele reînnoite vor fi utilizate în planul de optimizare a interogărilor. Setarea acestei opţiuni cu **True** nu va avea nici un efect, dacă pentru ***Auto Update Statistics***, de asemenea, nu va fi setată valoarea **True**.

Containment

Secţiunea data devine disponibilă pentru editare numai în cazul în care bazele de date sunt de tip *izolat* (*Contained*). Bazele de date izolate nu conţin dependenţe de configuraţie din SQL Server şi includ toate metadatele necesare direct în corpul sau. Această proprietate permite deplasarea uşoară a bazelor de date de la o instanţă SQL Server la alta.

- ***Default Fulltext Language LCID*** – specifică limba implicită pentru coloanele full-text indexate.
- ***Default Language*** – specifică limba implicită pentru toate loghinurile noi create. Limba poate fi specificată prin indicarea Id-ului local (lcid), denumirii limbii sau aliasului limbii.
- ***Nested Triggers Enabled*** – specifică dacă un declanşator *AFTER* poate sa se declanşeze in cascadă, adică execută o acţiune, care iniţiază un alt declanşator şi aşa mai departe.
- ***Transform Noise Words*** – opţiunea este utilizată pentru suprimarea mesajelor de eroare dacă operaţia booleană pe o interogare full-text eşuează din cauza cuvintelor "zgomot" sau „stopwords".
- ***Two Digit Year Cutoff*** – specifică un număr întreg între 1753 şi 9999 care reprezintă valoarea-prag pentru transformarea anilor din 2 cifre in 4.

Cursor

- ***Close Cursor on Commit Enabled*** – specifică dacă cursoarele se închid după validarea tranzacţiei care a deschis a cursorul. Valorile posibile sunt **True** sau **False.** Dacă se indică valoarea **True,** orice cursor deschis este închis, dacă tranzacţia este validată (*COMMIT*) sau tranzacţia este desfăşurată înapoi (*ROLLBACK*). Dacă este indicată valoarea **False**, cursoarele rămân deschise după validarea tranzacţiei. În cazul când valoarea este **False** şi tranzacţia este desfăşurată înapoi, se închid toate cursoarele cu excepţia celor definite cu clauzele *INSENSITIVE* sau *STATIC*.
- ***Default Cursor*** - specifică comportamentul implicit al cursorului. Dacă se atribuie valoarea **True,** valoarea implicită a cursorului este *LOCAL*. În caz contrar, este *GLOBAL*.

Filestream

- ***Filestream Directory Name*** – specifică numele dosarului pentru datele *FILESTREAM* asociate cu baza de date selectată.
- ***Filestream Non-transacted Access*** – specifică una din opţiunile următoare pentru accesul netranzacţional, prin sistemul de fişiere, la datele *FILESTREAM* stocate în tabelele de tip ***FileTables***: *OFF*, *READ_ONLY* sau *FULL*. Dacă tipul de date *FILESTREAM* nu este activat pe server, această valoare este setată la *OFF* şi este dezactivată.

Miscellaneous

- ***Allow Snapshot isolation*** – poate lua valorile *ON* sau *OFF*. Daca opţiunea este activată (*ON*), instrucţiunile DML încep a genera versiunile înregistrărilor chiar dacă nici o tranzacţie nu utilizează izolarea *SNAPSHOT*. În cazul în care tranzacţia se execută la nivelul de izolare *SNAPSHOT*, toate instrucţiunile văd snapshot-urile datelor aşa cum el existau la începutul tranzacţiei.
- ***ANSI Null Default*** - indică comportamentul implicit al operatorilor de comparaţie logică egal (=) şi neegal (<>) în cazul utilizării valorii *NULL*. Valorile posibile sunt **True** (on) şi **False** (off). Pentru mai multă informare, vezi *SET ANSI_NULLS* (Transact-SQL).
- ***ANSI Nulls Enabled*** - indică comportamentul operatorilor de comparaţie logică egal (=) şi neegal (<>) în cazul utilizării valorii *NULL*. Valorile posibile sunt **True** (on) şi **False** (off). Dacă se indică valoarea **True,** toate comparaţiile cu valoarea *NULL* se evaluează *UNKNOWN*. Dacă se indică valoarea **False**, comparaţiile valorilor non-*UNICODE* cu o valoare *NULL* se evaluează cu **True,** dacă ambele valori sunt *NULL*.
- ***ANSI Padding Enabled*** – specifică dacă formatul ANSI este activat sau dezactivat. Valorile posibile sunt **True** (on) şi **False** (off). Pentru mai multă informare, vezi *SET ANSI_PADDING* (Transact-SQL).
- ***ANSI Warnings Enabled*** – specifică comportamentul conform standardului ISO pentru mai multe condiţii de eroare. Dacă se indică valoarea **True,** este generat un mesaj de avertizare, în cazul când valorile *NULL* apar în funcţiile de agregare (precum *SUM*, *AVG*, *MAX*, *MIN*, *STDEV*, *STDEVP*, *VAR*, *VARP* sau *COUNT*). Dacă se indică valoarea **False,** nu este afişat nici un avertisment.
- ***Arithmetic Abort Enabled*** – specifică dacă opţiunea bazei de date pentru întreruperi aritmetice este activată sau nu. Valorile posibile sunt **True** şi **False.** Dacă se indică valoarea **True,** o supraîncărcare sau împărţire la zero cauzează terminarea unei cereri sau unei mulţimi de cereri. Dacă se indică valoarea **False,** este afişat un mesaj de avertizare dar interogarea, mulţimea de cereri sau tranzacţia continuă, dacă nu apare nici o greşeală.
- ***Concatenate Null Yields Null*** – specifică comportamentul când sunt concatenate valorile *NULL*. Dacă valoarea proprietăţii este **True**, atunci ***string*** + *NULL* va returna *NULL*. Dacă se indică valoarea **False,** rezultatul va fi ***string***.
- ***Cross-database Ownership Chaining Enabled*** - această valoare nu este editabilă şi indică faptul că dacă a fost sau nu activată înlănţuirea interbază de proprietate. Această

proprietate poate fi modificată, utilizând instrucţiunea *ALTER DATABASE*.

- ***Date Correlation Optimization Enabled*** – Dacă este selectată valoarea **True**, SQL Server menţine statisticile de corelare între oricare două tabele din baza de date care sunt legate printr-o constrângere *FOREIGN KEY* şi au coloane de tip *DATETIME*. Dacă - **False**, statisticile de corelare nu sunt întreţinute.
- ***Delayed durability*** - verifică dacă tranzacţiile validate sunt complet-durabil sau întârziat-durabil. Această opţiune poate lua valori *DIABLED, ALLOWED, FORCED*.
- ***Is Read Committed Snapshot*** - dacă această opţiune este activată, pentru obţinerea datelor consistente la citire, tranzacţiile utilizează la citire versiunile liniilor în loc de mecanismele de blocare a datelor. Această opţiune nu poate fi activată asupra bazelor de date de sistem.
- ***Numeric Round-Abort*** – specifică modul de gestiune a erorilor de rotunjire. Valorile posibile sunt **True** şi **False.** Dacă valoarea proprietăţii este **True**, este generată o eroare atunci când, într-o expresie, apare pierderea de precizie. Dacă se indică valoarea **False,** pierderea preciziei nu implică generarea unui mesaj de eroare şi rezultatul este rotunjit după precizia coloanei sau variabilei ce păstrează rezultatul.
- ***Parameterization*** – dacă este activ parametrul *SIMPLE*, interogările sunt parametrizate bazându-se pe comportamentul implicit al bazei de date. Dacă, însă, e activ parametrul *FORCED*, SQL Server parametrizează toate interogările asupra bazei de date.
- ***Quoted Identifiers Enabled*** – specifică dacă cuvintele-cheie ale sistemului ***SQL Server*** pot fi utilizate în calitate de identificatoare (obiect sau nume de variabilă), dacă sunt luate între ghilimele.
- ***Recursive Triggers Enabled*** – specifică dacă declanşatoarele pot fi lansate de alte declanşatoare. Valorile posibile sunt **True** şi **False.** Dacă este setată valoarea **True**, este activat apelul recursiv al declanşatoarelor. Dacă este setată valoarea **False,** este evitată numai recursia directă. Pentru a dezactiva recursia indirectă, se setează opţiunea **nested triggers** cu valoarea 0 folosind ***sp_configure***.
- ***Trustworthy*** – în cazul afişării valorii **True**, această opţiune read-only indică faptul că SQL Server permite accesul la resursele din afara bazei de date în cadrul unui context impersonal stabilit în baza de date. Contextul impersonal poate fi stabilit în baza de date, folosind instrucţiunea *EXECUTE AS*. Implicit în toate bazele de date, inclusiv şi în cele de sistem (excepţie face baza de date MSDB) acesta opţiune este dezactivată – **False**.
- ***VarDecimal Storage Format Enabled*** – acesta opţiune este setată la *read-only* începând cu versiunea SQL Server 2008. Valoare ***True*** indică faptul că baza de date are formatul de stocare *VarDecimal* activat. Formatul de stocare *VarDecimal* nu poate fi dezactivat atâta timp cât cel puţin un tabel din baza foloseşte acest format. În SQL Server 2008 şi versiunile ulterioare, toate bazele de date sunt activate pentru formatul de stocare *VarDecimal*.

Recovery

- ***Page Verify*** – specifică opţiunea utilizată pentru descoperirea şi semnalarea tranzacţiilor de intrare/ieşire incomplete cauzate de erorile de intrare/ieşire ale discului. Valorile posibile sunt ***NONE, TORN_PAGE_DETECTION*** şi ***CHECKSUM***.
- ***Target Recovery Time (Seconds)*** – specifică limita de timp maximală, exprimată în secunde, pentru recuperarea bazei de date specificate în cazul unei căderi.

Service Broker

- ***Broker enabled*** – poate lua valorile **True** sau **Fals**. **True** indică faptul că ***Service Broker*** este activat pentru baza de date în uz şi livrarea mesajelor este pornită. Baza de date păstrează identificatorul existent al serviciului ***Service Broker***.

- ***Honor Broker Priority*** – valorile disponibile pentru această opţiune sunt *ON* şi *OFF*. Valoarea *ON* indică sistemului să ia în considerare nivelurile de prioritare care sunt atribuite conversaţiilor. Mesajele de la conversaţiile care au un nivel de prioritate mai înalt sunt trimise înaintea mesajelor de la conversaţiile cu un nivel mai jos de prioritate. Valoare *OFF* – indică faptul că toate conversaţiile intr-o bază de date şi aplicaţiile posedă nivelul de prioritate implicit.

State

- ***Database Read Only*** – specifică dacă baza de date este numai pentru citire. Valorile posibile sunt **True** şi **False.** Dacă este setată valoarea **True**, utilizatorii pot doar citi datele din baza de date. Utilizatorii nu pot modifica datele sau obiectele bazei de date. Însăşi baza de date poate fi ştearsă utilizând instrucţiunea *DROP DATABASE*. Baza de date nu poate fi în uz atunci când este specificată o valoare nouă pentru opţiunea ***Database Read Only.*** Baza de date **master** este o excepţie, dar numai administratorul de sistem poate utiliza această bază în timpul când se setează opţiunea.
- ***Database State*** – vederea stării curente a bazei de date. Nu este editabilă. Pentru mai multă informare, vezi ***Database State.***
- ***Encryption Enabled*** – dacă este setată valoarea **True**, baza de date este activată pentru criptare. Pentru criptare este necesară o cheie de criptare.
- ***Restrict Access*** – specifică utilizatorii care pot accesa baza de date. Valorile posibile sunt:
  - ***Multiple*** – starea normală a bazei de date care permite multor utilizatori să acceseze baza de date în acelaşi timp.
  - ***Single*** – se utilizează pentru acţiunile de menţinere a bazei şi permite accesul la aceasta numai unui singur utilizator într-un moment de timp.
  - ***Restricted*** – pot utiliza baza de date numai membrii rolurilor ***db_owner***, ***dbcreator*** sau ***sysadmin***.

Pentru schimbarea valorii opţiunii respective, se face un click în coloana de valori, se alege valoarea nouă din meniul de context apărut sau executând un dublu click pe opţiunea respectivă.

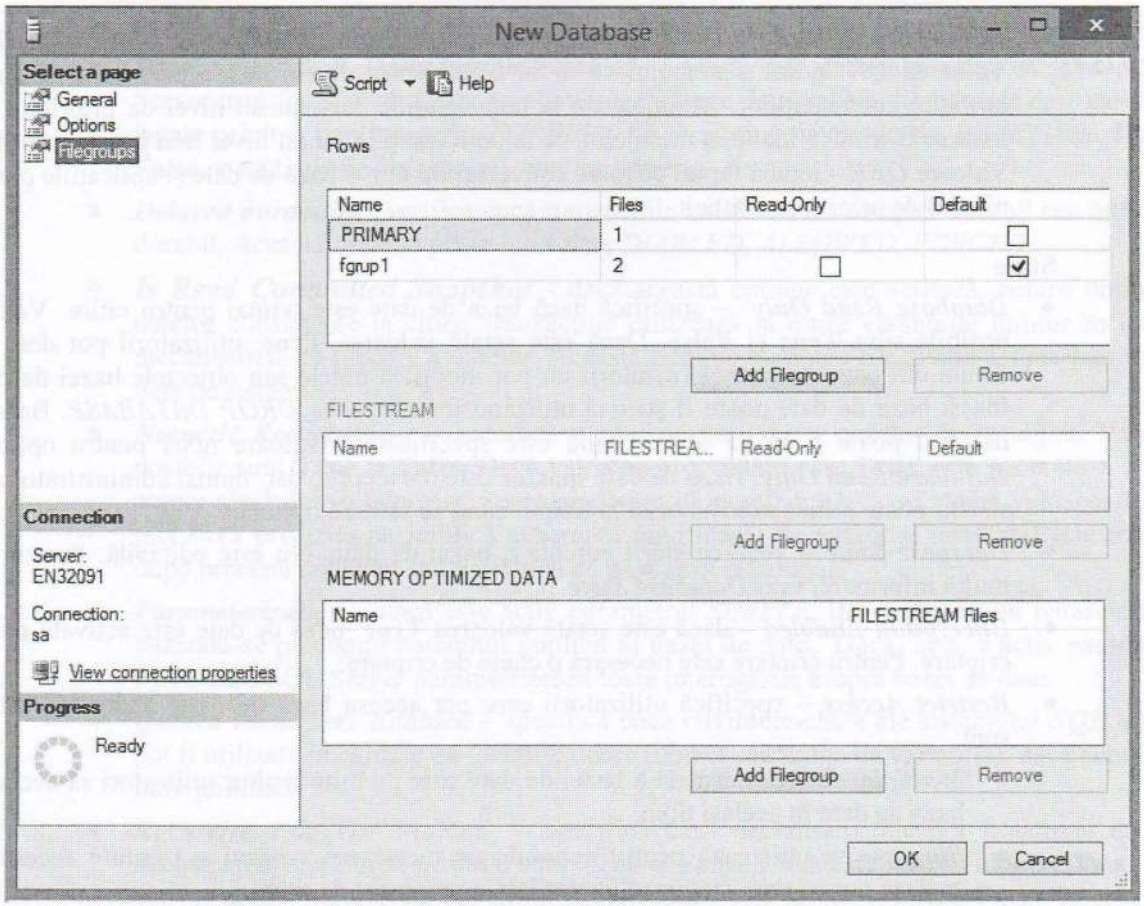
Unele proprietăţi suplimentare ale bazelor de date pot fi setate numai cu limbajul Transact-SQL în cadrul instrucţiunii *ALTER DATABASE.*

2.2.4. Afişarea şi adăugarea grupurilor de fişiere

Ultima pagină ***Filegroups*** – se utilizează pentru vizualizarea grupurilor de fişiere sau adăugarea grupurilor noi în baza de date selectată. Este alcătuită din 2 panele editabile, ***Rows*** şi ***Filestream.***

Panelul ***Rows*** cuprinde următoarele funcţionalităţi:

- ***Name*** – introduce numele grupului nou de fişiere.
- ***Files*** – afişează numărul de fişiere în grupul de fişiere.
- ***Read-only*** – grupului de fişiere i se oferă doar statutul de citire.
- ***Default*** – grupul de fişiere este declarat grup de fişiere implicit.
- Butonul **Add** – adaugă, pentru baza de date curentă, un rând nou la tabelul grupurilor de fişiere.
- Butonul **Remove** – şterge rândurile selectate de grupuri de fişiere din tabel.



11. Panelul ***Filestream*** este utilizat pentru gestiunea grupurilor de fişiere ce conţin obiecte BLOB. Utilizarea tehnologiei *Filestream* depăşeşte limitele acestei lucrări.
12. Panelul de gestiune a grupurilor de fişiere ***Memory Optimized Data*** – permite gestiunea fişierelor asociate în SQL Server, adăugarea/excluderea fişierelor şi grupurilor de fişiere din baza de date şi modificarea atributelor bazei de date sau fişierelor ei.

    Name – introduce numele grupului nou de fişiere memory-optimized.

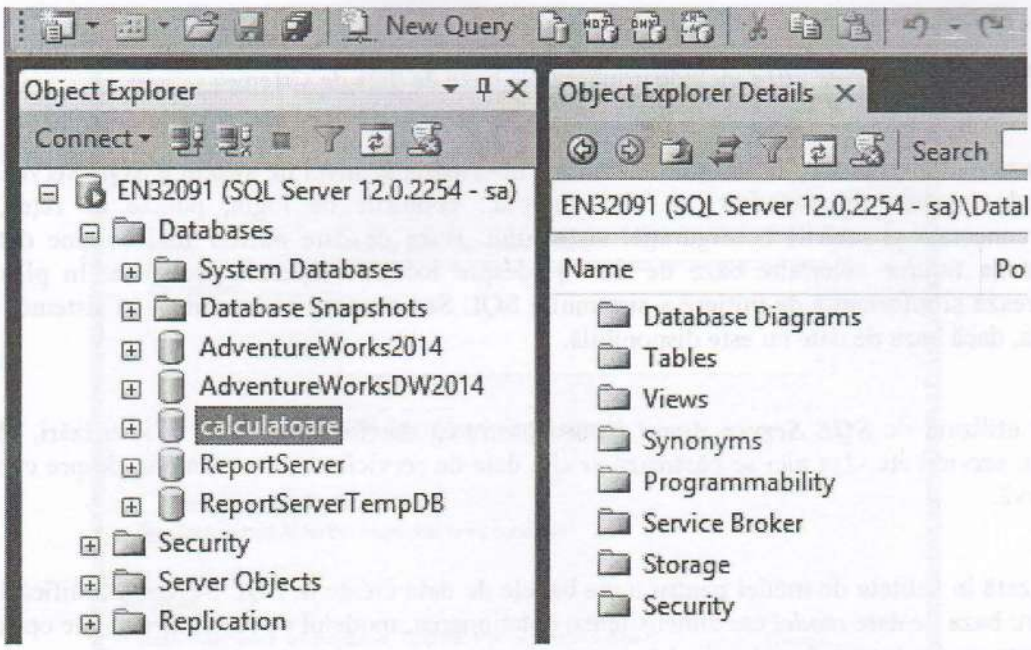
    Filestream Files – afişează numărul de fişiere (containere) în grupul de fişiere memory-optimized.

    Butonul ***Add*** – adaugă, pentru baza de date curentă, un rând nou la tabelul grupurilor de fişiere.

    Butonul ***Remove*** – şterge rândurile selectate de grupuri de fişiere din tabel.

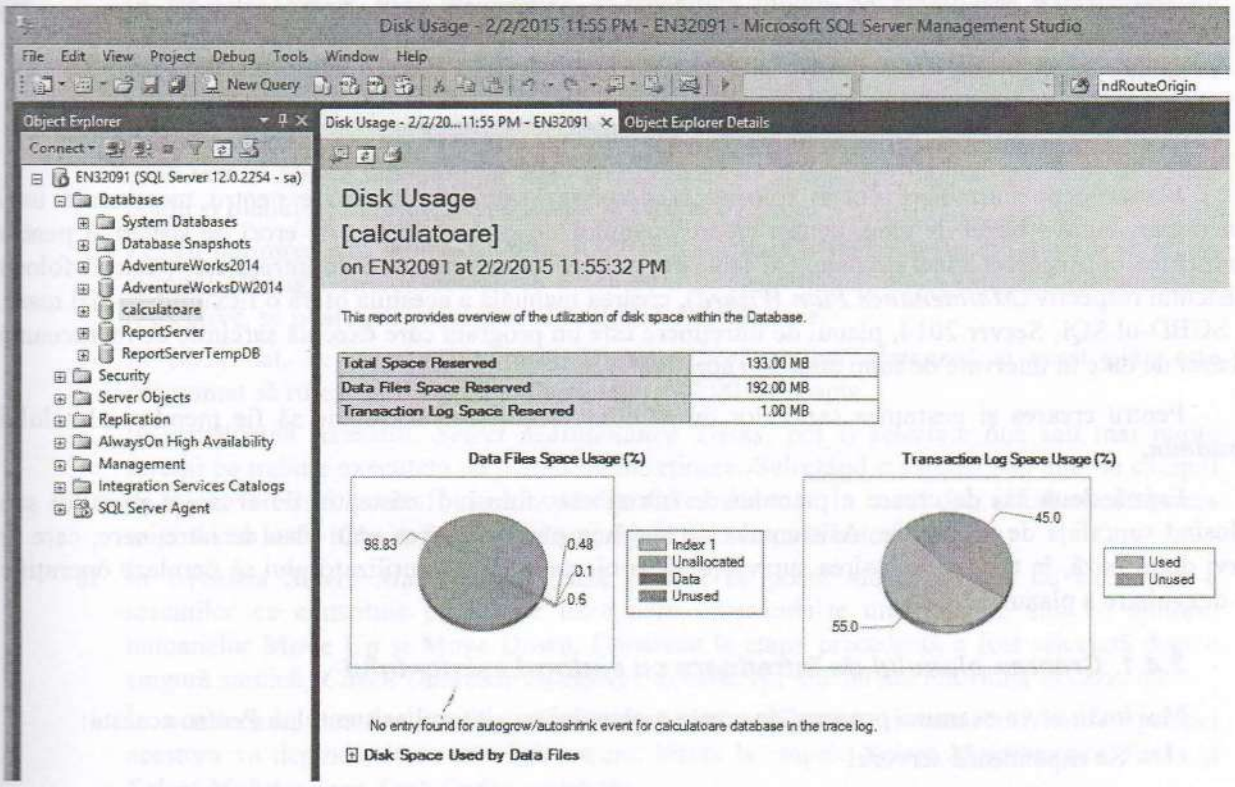
    După ce toate opţiunile bazei de date sunt definite, se apăsa butonul **OK**

    Se poate observa că baza de date recent creată, deja, se conţine în dosarul ***Databases***.



13. Cu scopul de a obţine informaţii statistice despre baza de date în uz se selectează baza de date respectivă, se apăsa butonul drept al mouse-ului şi din meniul de context se alege ***Reports,*** apoi se trece la ***Standard Reports*** şi din lista apărută se alege tipul raportului.

    De exemplu, în imaginea de mai jos a fost ales tipul de raport ***Disk Usage.*** În rezultat, au apărut date statistice despre utilizarea spaţiului de disc alocat fişierelor şi obiectelor bazei de date.



2.3. Baze de date de sistem

SGBD-ul ***SQL Server 2014*** include următoarele baze de date de sistem:

master

În baza de date ***master***, se înregistrează toată informaţia la nivel de sistem a SQL Server. Aceasta include instanţe de metadate, ca, de exemplu,: conturile de login, puncte de reţea, servere interconectate şi setările configuraţiei sistemului. Baza de date ***master*** mai conţine date despre existenţa tuturor celorlalte baze de date şi despre locaţiile fişierelor acestora. În plus, ***master*** păstrează şi informaţia de iniţiere a sistemului SQL Server, ceea ce înseamnă că sistemul nu poate starta, dacă baza de date nu este disponibilă.

msdb

Este utilizată de ***SQL Server Agent*** pentru păstrarea datelor referitoare la avertizări, planuri de lucru, servicii etc. Tot aici se păstrează şi alte date de serviciu ca, de exemplu, despre copierile de rezervă.

model

Utilizată în calitate de model pentru toate bazele de date create în SQL Server. Modificările făcute pentru baza de date ***model*** ca: dimensiunea, colaţionarea, modelul de restabilire şi alte opţiuni vor fi aplicate asupra tuturor bazelor de date create ulterior.

Resource

Este o bază de date numai pentru citire şi conţine obiectele sistemului incluse în SQL Server 2014. Obiectele sistemului persistă fizic în baza de date ***Resources***, dar, logic, acestea apar în schema ***sys*** a fiecărei baze de date. Acesta baza de date nu este vizibila in interfaţa sistemului.

tempdb

Baza de date de sistem ***tempdb*** este o sursă globală accesibilă tuturor utilizatorilor conectaţi la instanţa SQL Server şi conţine toate tabelele temporare şi procedurile stocate temporare. De asemenea, acoperă şi toate necesităţile unui spaţiu temporar de stocare a tabelelor de lucru generate de SQL Server.

2.4. Crearea planului de întreţinere

Planurile de întreţinere pot fi folosite la derularea sarcinilor necesare pentru menţinerea unei funcţionări bune a bazei de date, pentru crearea copiilor de rezervă în caz de erori de sistem şi pentru verificarea consistenţei bazei de date. Cu toate că, pentru crearea planului de întreţinere, poate fi folosit asistentul respectiv (***Maintenance Plan Wizard***), crearea manuală a acestuia oferă o flexibilitate mai mare. În SGBD-ul SQL Server 2014, planul de întreţinere este un program care execută sarcinile de mentenanţă a bazei de date în intervale de timp conform acestuia.

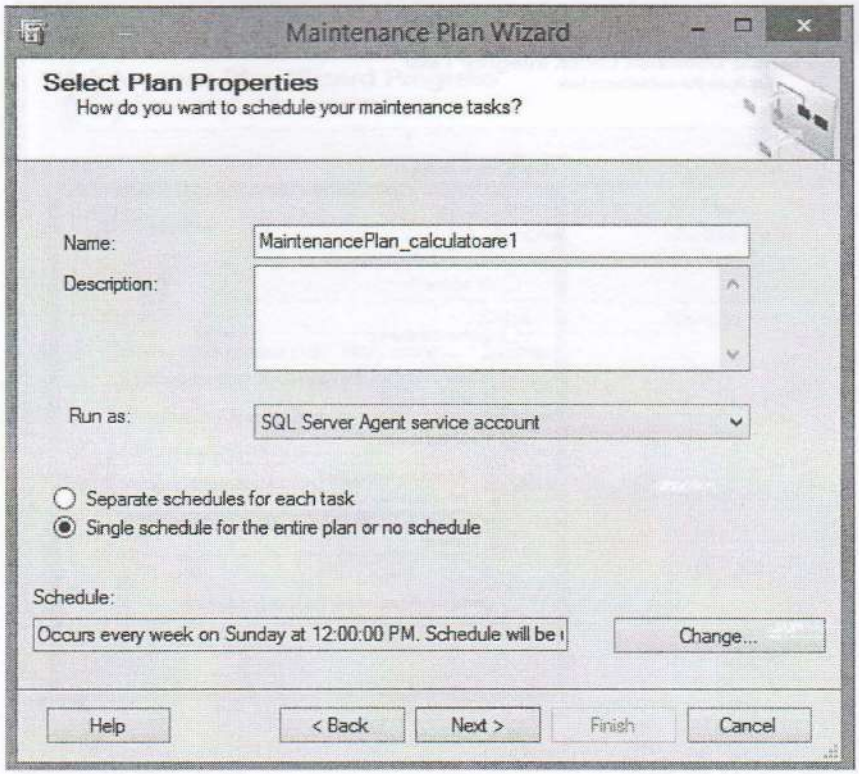
Pentru crearea şi gestiunea sarcinilor de menţinere, utilizatorul trebuie să fie membru al rolului ***sysadmin***.

Există două căi de creare a planului de întreţinere: folosind asistentul de creare a planului sau folosind suprafaţa de proiectare. Asistentul este mai bun pentru crearea unui plan de întreţinere, care ar servi drept bază, în timp ce folosirea suprafeţei de proiectare permite utilizatorului să deruleze operaţiile de dezvoltare a planului.

2.4.1. Crearea planului de întreţinere cu ajutorul asistentului

Mai întâi, se va examina procesul de creare a planului cu ajutorul asistentului. Pentru aceasta:

1. Se expandează serverul
2. Se expandează dosarul ***Management***
3. Se face un click pe butonul drept al mouse-ului asupra dosarului ***Maintenance plan***
4. Se selectează opţiunea ***Maintenance plan wizard***

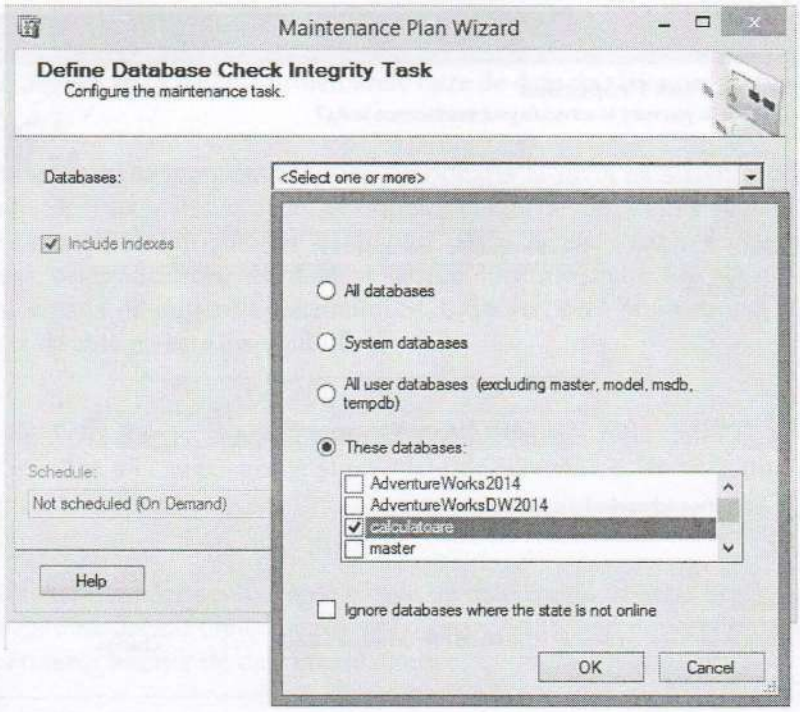


5. Textul afişat în prima fereastră a asistentului de creare a planului de mentenantă este unul introductiv şi prezintă un rezumat al sarcinilor, care pot fi automatizate de plan. Pentru a trece mai departe, se apasă butonul **Next.**
6. În fereastra ***Select Plan Propreties***, se specifică modul de planificare a sarcinilor de întreţinere. În câmpul ***Name,*** se specifică numele planului de întreţinere. În câmpul ***Decription,*** se pot adăuga comentariile referitoare la planul dat. Câmpul ***Run as*** indică contul din numele căruia va fi rulat planul de întreţinerea. Implicit, acesta este ***SQL Server Agent service account.*** Mai jos, sunt plasate două radiobutoane: ***Separate schedules for each task*** şi ***Single schedule for the entre plan or no schedule.*** Implicit este activ al doilea buton şi planul de întreţinere se execută la cerere.

   Pentru definirea sau modificarea orarului de executare a planului de întreţinere, se apasă butonul **Change**. În consecinţă, se deschide o fereastră nouă, unde, bifând radiobutoanele respective, se poate programa orarul dezirabil de executare.

   În cazul dat, se inserează numele ***MaintenancePlan_calculatoare1*** şi acest plan este programat să ruleze în fiecare duminică la ora 12:00 de noapte.
7. În următoarea fereastră, ***Select Maintenance Tasks***, pot fi selectate una sau mai multe sarcini ce trebuie executate de planul de întreţinere. Selectând o sarcină sau alta, în câmpul de jos al ferestrei, poate fi vizualizată descrierea acesteia. În cazul dat, a fost aleasă prima sarcină din listă - ***Check Database Integrity***. Apoi se apasă butonul **Next.**
8. În fereastra ***Select Maintenance Task Order***, se poate alege ordinea de executare a sarcinilor ce constituie planul de întreţinere, mişcându-le una faţă de alta cu ajutorul butoanelor **Move Up** şi **Move Down**. Deoarece la etapa precedentă a fost selectată doar o singură sarcină (***Check Database Integrity***), această opţiune nu are relevanţă în cazul dat.

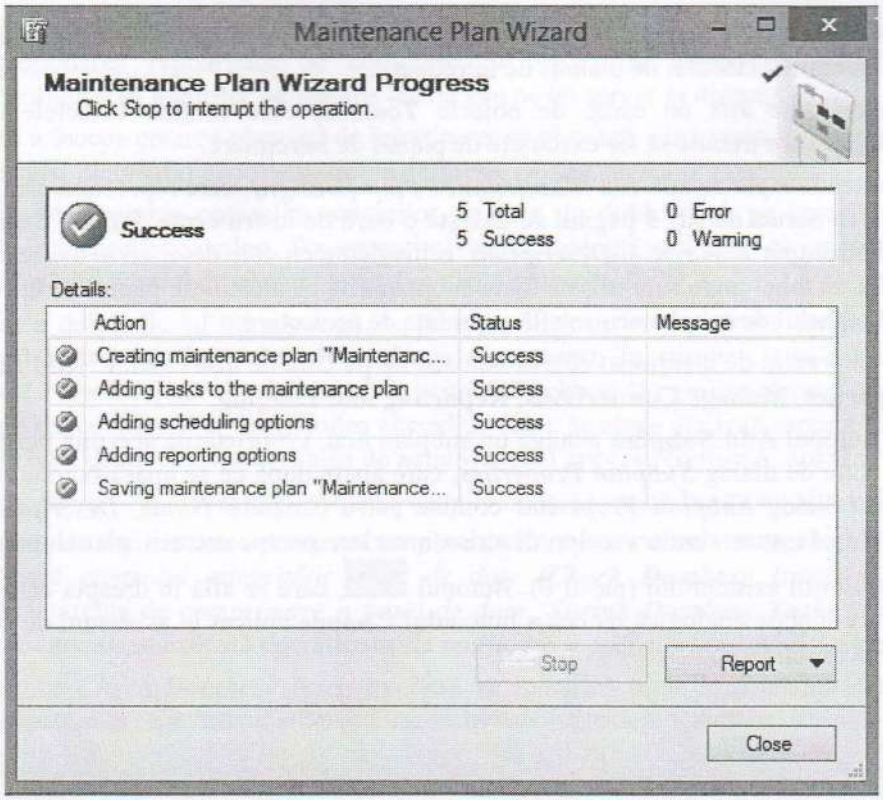
   În continuare, numărul paşilor (ferestrelor) de configurare şi consecutivitatea apariţiei acestora va depinde de numărul de sarcini bifate la etapele ***Select Maintenance Tasks*** şi ***Select Maintenance Task Order,*** respectiv.
9. În fereastra ***Define Database Check Integrity Task***, pot fi indicate bazele de date care vor fi întreţinute automat de asistent. Executând un click pe săgeată în meniul ***Databases,*** apare o fereastră în care poate fi aleasă şi marcată una din opţiunile ce urmează:



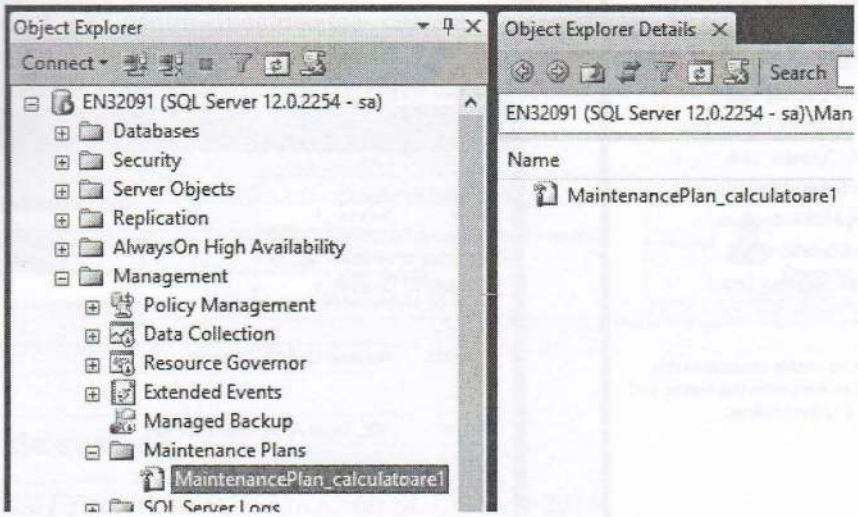
- toate bazele de date;
- numai bazele de date ale sistemului;
- bazele de date create de utilizator;
- bazele de date selectate din listă.

În cazul dat, este selectată opţiunea ***These database*** şi este aleasă doar baza de date recent creată, după care se aplică un click pe **OK** şi în fereastra principală - **Next.**

10. La etapa ***Select Report Options***, asistentul vrea să cunoască dacă raportul planului de întreţinere trebuie să fie plasat într-un dosar pe serverul care realizează întreţinerea sau trebuie trimis pe *e-mail*ul unui operator. Implicit este bifată opţiunea de plasare a raportului într-un dosar (***Write a report to a text file***). În câmpul ***Folder location,*** se inserează calea unde va fi salvat fişierul-raport. Direcţia salvării fişierului poate fi editată manual sau apăsând butonul **Browse,** ....
11. În fereastra ***Complete the Wizard*** este prezentat sub formă de arbore un rezumat al operaţiilor care vor fi realizate de asistent. Pentru vizualizarea detaliilor, se expandează opţiunea ce ne interesează, apăsând semnul plus respectiv. Pentru a termina procesul de proiectare a planului de întreţinere, se face un click pe butonul **Finish**.
12. În fereastra ***Maintenance Plan Wizard Progress*** are loc procesarea sarcinii de creare a planului de întreţinere. În panelul ***Details*** se vizualizează lista de acţiuni, statutul acestora şi mesajele în cazul apariţiei unui eşec de procesare. Apăsând butonul **Report**, se poate vizualiza, salva sau expedia prin *e-mail* raportul de procesare a planului de întreţinere. Dacă statutul pentru toate acţiunile este ***Success***, se apasă butonul **Close**.



13. Acum, în ***SQL Server Management Studio***, se poate verifica dacă planul de întreținere a fost creat. În dosarul ***Maintenance Plan,*** trebuie să fie prezent un plan de mentenanță nou cu numele respectiv.



2.4.2. Crearea manuală a planului de întreținere

Planul de întreținere poate fi creat utilizând și regimul constructorului. Pentru aceasta:

1. Se expandează serverul.
2. Se expandează dosarul ***Management.***
3. Se face un click pe butonul drept al mausului asupra dosarului ***Maintenance Plans.***
4. Se selectează opțiunea ***New Maintenance plan…***
5. Apare o cutie de dialog, unde se scrie numele planului de întreținere ce se creează.

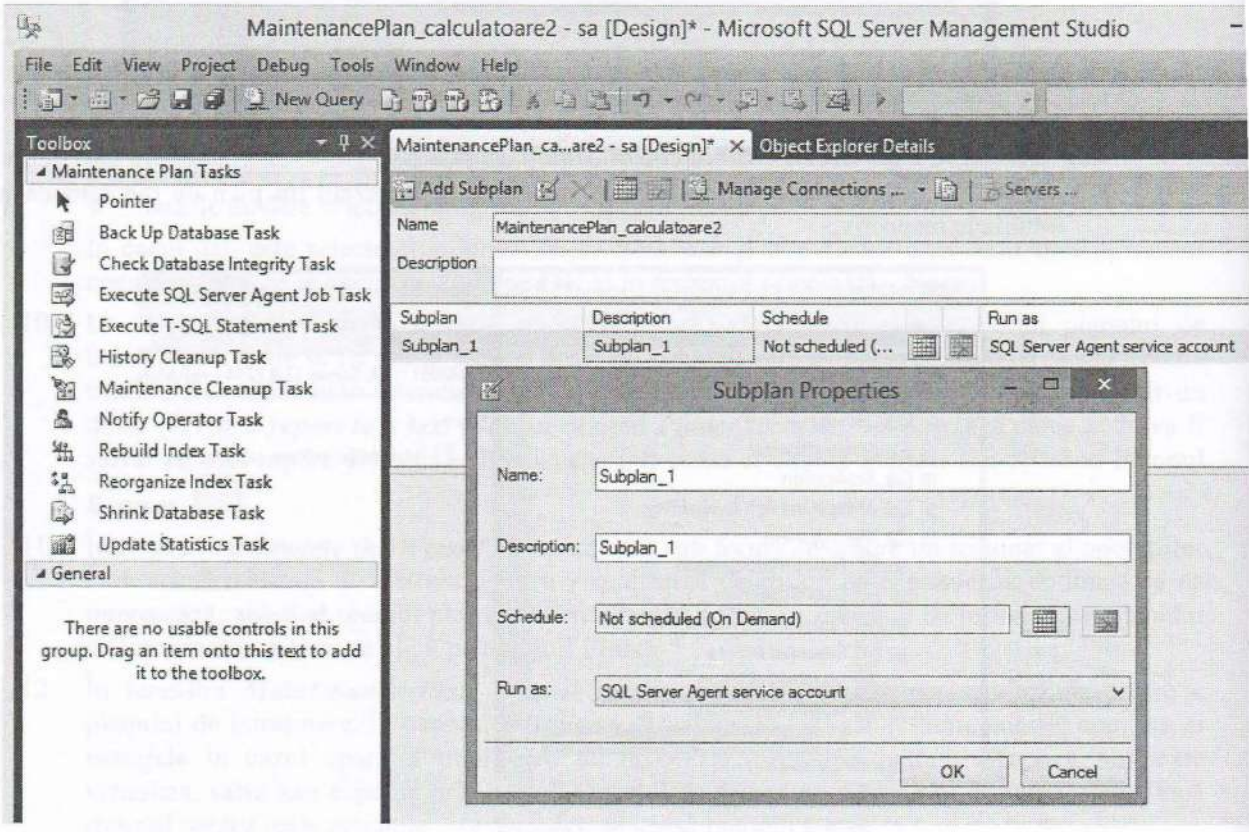
După ce se apasă butonul **OK**, în ***Management Studio*** se deschide o filă nouă ce reprezintă mediul constructorului de planuri de întreținere.

În stânga, se află un câmp de obiecte ***Toolbox,*** care conține obiectele corespunzătoare sarcinilor care trebuie să fie executate de planul de întreținere.

În partea dreaptă se află fila ***Maintenance Plan [Design]***, care reprezintă câmpul principal de lucru. În partea de sus a paginii se găsește o bară de instrumente, iar sub bara de instrumente se află câmpurile ***Name*** și ***Description***, echivalente cu cele descrise în asistent. Puțin mai jos, se află un tabel unde sunt afișate toate subplanurile ce alcătuiesc planul de întreținere dat. Mai jos de tabelul de subplanuri, se află suprafața de proiectare.

Este important de menționat câteva butoane de pe bara de instrumente: **Add Suplan**, **Subplan Properties**, **Manage Connections, Reporting and Logging**.

- Butonul **Add Subplan** adaugă un subplan nou. Proprietățile acestuia pot fi configurate în cutia de dialog ***Subplan Properties,*** care apare după ce se apasă butonul respectiv. Cutia de dialog ***Subplan Properties*** conține patru câmpuri: ***Name, Description, Schedule*** și ***RunAs*** sunt similare celor descrise anterior, pentru crearea planului de întreținere cu ajutorul asistentului (pasul 6). Butonul , care se află în dreapta câmpului ***Schedule,*** are funcție analogică cu cea a butonului **Change** aplicat în asistentul de creare a planului de întreținere.



- Al doilea buton din dreapta, **Subplan Properties**, lansează cutia de dialog cu același nume. Selectând subplanul din tabelul de subplanuri de mai jos și apăsând acest buton, se pot insera numele, descrierea și se poate planifica un subplan în timp. Cutia de dialog ***Subplan Properties*** poate fi vizualizată, aplicând un dublu click pe un subplan din tabel.
- Butonul **Manage Connections** afișează cutia de dialog respectivă. Aici pot fi gestionate conexiunile utilizate in planul de întreținere. Se folosește pentru adăugarea altor instanțe ale SQL Server conectate la planul de întreținere în cauza. Fiecare sarcină de întreținere din subplanul dat poate utiliza oricare din aceste conexiuni.

- Butonul **Reporting and Logging** afişează cutia de dialog cu acelaşi nume, utilizată pentru gestionarea rapoartelor de activitate cu privire la planul de întreţinere şi pentru configurarea logurilor pe maşina locală sau pe un server la distanţă

6. Pentru a începe crearea planului de întreţinere, se glisează, din caseta **Toolbox,** pe suprafaţa de proiectare, elementul necesar pentru a defini sarcinile ce trebuie executate.
7. Pentru configurarea opţiunilor sarcinilor, se face un dublu-click pe sarcina respectivă şi se deschide o casetă de dialog. Tot aceeaşi casetă de dialog apare după ce se dă un click-drept pe sarcină şi din meniul de context apărut se alege opţiunea ***Edit.***
8. Pentru a defini fluxul între sarcini, se face un click-drept pe oricare sarcină şi în meniul de context apărut se alege ***Add Precedence Constraint.*** În rezultat, apare o fereastră numită ***Conrol Flow*** cu două check boxuri. În primul checkbox (***From***) se alege din listă sarcina care trebuie executată prima. În al doilea checkbox (***To***), se alege din listă sarcina care trebuie să fie executată a doua. Consecutivitatea de acţiuni aleasă apoi, se confirmă, apăsând butonul **OK**.

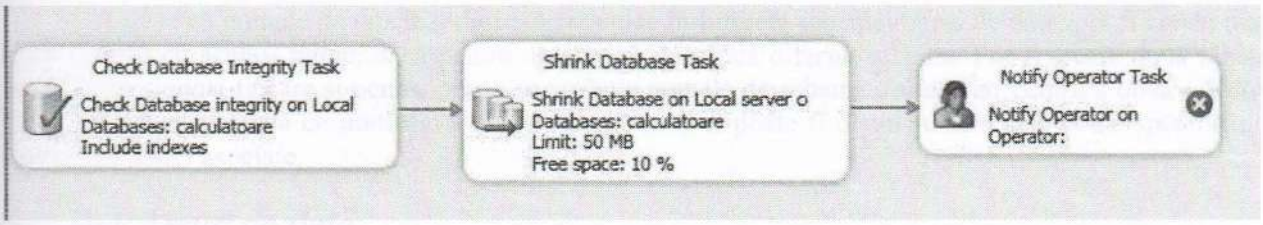
   Acelaşi lucru poate fi realizat, trăgând săgeţile de la o sarcină la alta cu ajutorul mouse-ului.

   În figura de mai jos, este prezentat un plan de întreţinere creat din trei sarcini. Prima sarcină realizează *controlul integrităţii bazei de date* ***(Check Database Integrity Task)***, apoi se execută sarcina de *comprimare a bazei de date* ***(Shrink Database Task)*** şi după executarea acestor sarcini, se execută operaţiunea de *notificare a operatorului* ***(Notify Operator Task).***

   În sarcina ***Check Database Integrity Task*** se specifică tipul de conexiune, numele şi parola utilizatorului şi se selectează baza de date. În caz concret, a fost ales serverul local, tipul *Mixt* de autentificare cu loginul ***sa*** şi parola respectivă şi a fost selectată baza de date *calculatoare*.

   Pentru sarcina ***Shrink Database Task,*** în ce priveşte conexiunea şi baza de date, setările sunt similare cu cele din sarcina precedentă. Restul parametrilor specifici operaţiunii de comprimare a bazei de date au luat valorile implicite.

   Dacă, în procesul de validare, se determină că o sarcină conţine unele erori, în partea dreaptă a pictogramei, apare un semn roşu, precum se vede la sarcina ***Notify Operator Task***. Trebuie menţionat că, pentru funcţionarea corectă a operaţiunii ***Notify Operator,*** este necesară configurarea serviciului ***Database Mail,*** lucru ce depăşeşte limitele volumului dat. După corectarea erorilor, se apasă butonul ***Save all*** de pe bara de instrumente.



Întrebări de control:

1. Tipuri de fişiere utilizate de Microsoft SQL Server 2014
2. Structura paginilor de păstrare a datelor.
3. Baze de date de sistem, importanţa şi rolul lor.
4. Principiile de creare a bazei de date în mediul ***SQL Server Management Studio***.
5. Ce este un plan de întreţinere a bazei de date şi care sunt tehnicile de creare a planului de întreţinere?

Sarcini practice:

1. Creaţi o bază de date plasată fizic în mapa ***MyDocuments\Data***, fixând o creştere a fişierului primar a bazei de 5MB cu limita de creştere de 100 MB şi a *log*ului de 20 MB cu limita de creştere de 1000 MB. Pentru fişierele secundare să se definească un ***Filegroup*** nou implicit, setând creşterea fişierelor secundare de 10 MB cu limita de 1000 MB.

2. Creaţi o bază de date, unde fişierul *log* să fie fizic plasat în mapa ***MyDocuments\Log***, numele fişierului *log* în mediul sistemului de operare trebuie să se deosebească de cel logic definit în schema fizică. Este important ca baza de date creată să fie compatibilă cu sistemul *MS SQL Server 2014* şi ea să fie accesibilă numai unui singur utilizator într-un moment de timp.
3. Creaţi planul de întreţinere a bazei de date, construită în sarcina 1. Spaţiul neutilizat de fişierele bazei de date trebuie îndepărtat atunci când el atinge mărimea 2000Mb. Spaţiul eliberat trebuie să fie returnat sistemului de operare. Această operaţiune trebuie să ruleze în fiecare vineri, la ora 02:00. Raportul executării planului de întreţinere trebuie salvat în docarul ***MyDocuments\SQL_reports***. Iniţializaţi executarea planului. După executare, verificaţi rezultatele în fişierul *log*.
4. Creaţi planul de întreţinere a bazei de date, construite în exerciţiul 2. Numele planului va fi: „***Reconstruire index***". În cadrul acestui plan, sistemul trebuie să realizeze reconstruirea indecşilor numai asupra tabelelor de bază (exclusiv viziunilor) din toate schemele care există în baza de date în cauză. Spaţiul liber pe pagină trebuie să fie 10%. După reconstruire, trebuie să urmeze colectarea statisticilor complete despre indecşii reconstruiţi. Al treilea pas al planului trebuie să constituie sarcina de ştergere a istoriei despre operaţiile de Backup-Restore ce au avut loc pe SQL Server. Trebuie şters istoricul care este mai vechi de 6 săptămâni. Acest plan trebuie să fie executat în fiecare prima duminică a lunii. Creaţi dosarul ***MyDocuments\SQL_reports.*** Raportul de executare a planului trebuie să fie adăugat în acest fişier. Iniţializaţi executarea planului. După executare, verificaţi rezultatele în fişierul *log* generat.

3. CREAREA ŞI MODIFICAREA TABELELOR ÎN SQL SERVER MANAGEMENT STUDIO

Capitolul tratează problemele cu privire la crearea tabelelor, definirea coloanelor şi setarea proprietăţilor de bază ale acestora, setarea proprietăţilor extinse ale coloanelor, stabilirea cheilor primare şi celorlalte constrângeri de integritate, precum şi modificarea structurii unui tabel. Sunt examinate, de asemenea, tipurile de date, constrângerile de integritate şi regulile de modificare a structurii unui tabel [6, 7].

Înainte de a trece la crearea tabelelor, se presupune că a fost făcut un studiu minuţios al datelor ce trebuie stocate. Mulţi elaboratori amatori au obiceiul să proiecteze baza de date atunci când, deja, se găseşte înăuntrul modulului de creare a tabelelor, fără a face un studiu şi o definire prealabilă a datelor necesare pentru construirea acestora. O astfel de atitudine, deseori, este însoţită de un risc, în ceea ce priveşte calitatea datelor.

SGBD-ul SQL Server are de-a face cu două tipuri de tabele:

1. Temporare, create în memoria volatilă.
2. Permanente, create şi salvate pe disc.

Fiecare tabel creat de SQL Server poate avea cel mult 1024 de câmpuri (coloane). Fiecare câmp (coloană) trebuie să posede unele specificaţii, cum sunt numele câmpului, tipul de date pe care le va păstra acest câmp. Unele informaţii sunt opţionale. De exemplu, valoarea implicită a câmpului sau unele limitări ale valorii, cum este valoarea maximă etc.

Numărul de înregistrări pe care le poate avea un singur tabel este limitat de locul (volumul de memorie) de păstrare a acestuia.

La fel ca şi în alte SGBD-uri, în SQL Server, fiecare tabel are un nume şi acest nume este unic, precum au nume diferite şi câmpurile unui tabel. De exemplu, în acelaşi tabel, nu pot fi două câmpuri *Telefon*. În asemenea caz, pot fi create două câmpuri, de exemplu, *Telefon_serviciu* şi *Telefon_domiciliu*.

Cu toate că numele de tabele trebuie să fie unice în limitele schemei bazei de date, pot fi create mai multe tabele cu acelaşi nume, dacă pentru acestea se specifică diferite scheme. Pot fi create două tabele numite *funcţionari* fiecare şi pentru prima se atribuie numele de schemă ***Jonah***, iar pentru a doua - ***Sally***. Atunci când se lucrează cu unul din tabelele *funcţionari,* el poate fi distins dintre cele două, specificând numele schemei asociate.

3.1. Tipuri de date

Precum s-a menţionat anterior, tot câmpul (coloana) unui tabel trebuie să respecte tipul de date specificat pentru acest câmp. Această specificare se face la crearea tabelului. Sistemul SQL Server aduce cu sine o mulţime largă de tipuri de date. Cu toate acestea, în operaţiile specifice, utilizatorul poate crea propriile tipuri de date. În continuare, este prezentată o listă de tipuri de date pe care le susţine SGBD-ul SQL Server 2014.

3.1.1. Date numerice exacte

BIGINT

Păstrează numere cuprinse între -2^{63} şi $2^{63}-1$. Ocupă 8 octeţi.

INT

Păstrează numere cuprinse între -2^{31} (-2.147.483.648) şi $2^{31}-1$ (2.147.483.647). Ocupă 4 octeţi.

SMALLINT

Păstrează numere cuprinse între -2^{15} (-32768) şi $2^{15}-1$ (32767). Ocupă 2 octeţi.

TINYINT

Păstrează valori cuprinse între 0 şi 255. Ocupă 1 octet.

BIT

Un întreg cu valoarea 1 sau 0. Se foloseşte pentru a reprezenta *True* sau *False*, sau *Yes* sau *No.*

DECIMAL(P,S) sau ***NUMERIC(P,S)***

Pentru păstrarea numerelor zecimale exacte. Parametrul *p* indică numărul total de cifre pe care câmpul poate să le conţină. Parametrul *s* indică numărul de cifre după punctul zecimal. În SQL Server, tipul *NUMERIC* este un sinonim pentru tipul de date *DECIMAL*. În cazul în care este utilizată precizia maximă, valoarea validă este cuprinsă între $-10^{38}+1$ şi $10^{38}-1$.

MONEY

Păstrează valori de la -922.337.203.685.477,5808 la +922.337.203.685.477,5807. Ocupă 8 octeţi.

SMALLMONEY

Păstrează valori de la -214.748,3648 la 214.748,3647 şi ocupă 4 octeţi.

3.1.2. Date numerice aproximative

FLOAT(N)

Pentru păstrarea numerelor inexacte, cu zecimale periodice. În acest caz, datele sunt aproximative în acord cu procesorul sistemului. Parametrul *N* determină precizia numărului. Pentru *N* de la 1 la 24, precizia este de 7 cifre şi ocupă 4 octeţi. Pentru *N* de la 25 la 53, precizia este de 15 cifre şi ocupă 8 octeţi memorie.

REAL

Corespunde tipului *FLOAT(24)*

3.1.3. Date secvenţe de caractere

CHAR(N)

Pentru secvenţe de caractere cu lungime fixă (până la 8Ko)

VARCHAR[(*N* | *MAX*)]

Pentru secvenţe de caractere cu lungime variabilă (până la 8 Ko).

MAX indică faptul că mărimea maximă de memorie utilizată este 2^{31}-1 octeţi (2 Go). Dimensiunea de stocare este lungimea reală a datelor introduse + 2 octeţi.

TEXT

Pentru secvenţe de caractere ASCII cu lungime variabilă mai mare de 8 Ko (de exemplu, documente HTML), dar care nu depăşeşte valoarea 2^{31}-1 octeţi (2147483647 caractere).

3.1.4. Date binare

BINARY(N)

Utilizat pentru păstrarea numerelor hexazecimale cu lungime fixă (până la 8 Ko)

VARBINARY[(*N* | *MAX*)]

Utilizat pentru păstrarea numerelor hexazecimale cu lungime variabilă (până la 8 Ko)

MAX indică faptul că mărimea maximă de memorie utilizată este 2^{31}-1 octeţi (2 Go). Dimensiunea de stocare este lungimea reală a datelor introduse + 2 octeţi

IMAGE

Pentru date binare mai mari de 8 Ko, dar care nu depăşesc valoarea 2^{31}-1 octeţi. Este utilizat pentru păstrarea documentelor Word, tabelelor electronice Excel, imagini (fişiere bitmap, GIF şi JPEG)

3.1.5. Tipuri de date dată/oră

DATETIME

Păstrează date alfanumerice, care reprezintă data cuprinsă între 01/01/1753 şi 31/12/9999, şi timpul între 00:00:00 şi 23:59:59.997, inclusiv. Ocupă 8 octeţi memorie. Precizie 3.33 milisecunde.

DATETIME2

Reprezintă o extensie a tipului *DATETIME*, deoarece data cuprinde un interval mai larg şi timpul este redat cu o exactitate mai înaltă. Susţine definirea preciziei de reprezentare a părţii fracţionare de către utilizator.

Cuprinde valori între 01/01/0001 00:00:00.0000000 şi 31/12/9999 23:59:59.9999999, inclusiv. Ocupă 6 octeţi pentru partea fracţionară mai mică de 3; 7 octeţi pentru partea fracţionară 4 sau 5. Celelalte precizii necesită 8 octeţi memorie. Precizia - 100 nanosecunde.

DATE

Păstrează date alfanumerice, care reprezintă data cuprinsă între 01/01/0001 şi 31/12/9999 inclusiv. Ocupă 3 octeţi memorie. Precizie - o zi.

TIME

Păstrează date alfanumerice, care reprezintă timpul. Cuprinde valori intre 00:00:00.0000000 şi 23:59:59.9999999, inclusiv. Precizia părţii fracţionare poate varia între valorile 0 şi 7. Precizia implicită a părţii fracţionare este 7 şi ocupă 5 octeţi memorie. Precizia maximală e 100 nanosecunde.

DATETIMEOFFSET

Păstrează date alfanumerice, care, în afara de lună, an şi dată, mai reprezintă şi ora în formatul 1-24. Ora se racordează conform fusului orar setat pe calculator. Reprezintă data între 01/01/1753 şi 31/12/9999, inclusiv timpul între 00:00:00 şi 23:59:59.9999999, inclusiv. Ocupă 10 octeţi de memorie cu valoarea implicită a părţii fracţionare de 100 nanosecunde precizie.

SMALLDATETIME

Păstrează date alfanumerice, care reprezintă data şi timpul de la 1/1/1900 la 6/6/2079. Ocupă 4 octeţi de memorie. Precizie 1 minut.

3.1.6. Tipuri de date speciale

CURSOR

Un tip de date pentru variabile şi parametrii de ieşire ai procedurii stocate care conţine referinţă la un cursor. Orice variabilă creată cu tipul de date cursor poate lua valoarea *NULL*.

HIERARCHYID

Este un tip de date de sistem cu lungime variabilă. Se utilizează pentru crearea tabelelor cu structura ierarhică sau pentru referinţa la date cu structură ierarhică. Valorile de tip *HIERARCHYID* reprezintă poziţiile într-o ierarhie.

SQL_VARIANT

Un tip de date care stochează valori ale diferitelor tipuri de date suportate de SQL Server, cu excepţia tipurilor *TEXT*, *NTEXT*, *IMAGE*, *TIMESTAMP* şi *SQL_VARIANT*.

Tipul *SQL_VARIANT* poate fi utilizat pentru coloane, parametri, variabile valori returnate de funcţiile definite de utilizatori. Tipul *SQL_VARIANT* permite acestor obiecte ale bazei de date să susţină valori ale altor tipuri de date.

O coloană de tipul *SQL_VARIANT* poate conține rânduri de diferite tipuri. De exemplu, o coloană definită *SQL_VARIANT* poate conține valori de tipul *INT*, *BINARY* şi *CHAR*.

TABLE

Este un tip de date special, care poate fi utilizat pentru stocarea unui rezultat pentru tratarea ulterioară. Tipul *TABLE* este, în primul rând, utilizat pentru păstrarea unei mulțimi de rânduri obținute în calitate de rezultat al unei funcții cu valoarea unui tabel.

TIMESTAMP

Reprezintă un identificator unic pentru toată baza de date şi constituie un număr binar de 8Ko. Se utilizează în calitate de mecanism de marcare a versiunii unui rând din tabel, care se autoincrementează de fiecare dată când se actualizează un rând. Tipul *TIMESTAMP* este un sinonim pentru *ROWVERSION*.

UNIQUEIDENTIFIER

Un identificator unic global (GUID) care reprezintă un număr de 16 biți. Se foloseşte când o linie trebuie să fie unică între multe alte linii.

XML

Este tipul de date ce păstrează date *XML*. Instanțele *XML* pot fi păstrate într-o coloană sau într-o variabilă de tipul *XML*

3.1.7. Date UNICODE

Tradițional, tipurile de date *UNICODE* în SQL Server permit folosirea datelor, care sunt definite de un anumit set de caractere. Un set de caractere este ales în timpul instalării sistemului SQL Server şi rămâne neschimbat pe toată durata de funcționare a software-ului instalat. Folosind tipurile de date *UNICODE*, o coloană poate stoca toate caracterele care sunt definite în diferite seturi de caractere. Tipurile de date *UNICODE* ocupă de două ori mai mult spațiu de stocare decât tipurile de date non-*UNICODE*. Datele *UNICODE* utilizează setul de caractere *UNICODE UCS*-2.

NCHAR(N)

Secvență de caractere *UNICODE* de lungime fixă (până la 4000 de caractere).

NVARCHAR[(N | MAX)]

Secvență de caractere *UNICODE* de lungime variabilă (până la 4000 de caractere).

MAX indică faptul că mărimea maximă de memorie utilizată este 2^{31}-1 octeți (2 Go). Dimensiunea de depozitare, în octeți, este de două ori lungimea efectivă a datelor introduse + 2 octeți.

NTEXT

Secvență de caractere *UNICODE* de lungime variabilă, care nu depăşeşte valoarea 2^{30}.

Tipurile de date *NTEXT*, *TEXT* şi *IMAGE* vor fi eliminate în versiune viitoare a Microsoft SQL Server. Microsoft nu recomandă utilizarea acestor tipuri de date pentru dezvoltarea proiectelor noi. În schimb, este recomandată planificarea modificărilor aplicațiilor care folosesc tipuri de date menționate cu tipuri de date noi - *NVARCHAR (MAX)*, *VARCHAR (MAX)*, şi *VARBINARY (MAX)*.

3.1.8. Ordinea de precedență

Odată cu definirea tipurilor de date pentru câmpuri (coloane), utilizatorul este forțat să respecte tipul specificat. Aceasta este constrângerea asupra atributului. De exemplu, literele nu vor fi acceptate pentru câmpurile numerice.

Atunci când două expresii au tipuri de date cu diferite colațiuni, precizii, scale sau lungimi şi sunt îmbinate de un operator, caracteristicile rezultatului sunt determinate, precum urmează:

- Tipul de date al rezultatului este determinat ţinând cont de regulile de precedenţă a tipurilor de date din expresii. SQL Server 2014 utilizează ordinea de precedenţă pentru tipurile de date, precum urmează:

1. *USER-DEFINED DATA TYPES (*cea mai înaltă*)*
2. *SQL_VARIANT*
3. *XML*
4 *DATETIMEOFFSET*
5. *DATETIME2*
6. *DATETIME*
7. *SMALLDATETIME*
8 *DATE*
9 *TIME*
10. *FLOAT*
11. *REAL*
12. *DECIMAL*
13. *MONEY*
14. *SMALLMONEY*
15. *BIGINT*
16. *INT*
17. *SMALLINT*
18. *TINYINT*
19. *BIT*
20. *NTEXT*
21. *TEXT*
22. *IMAGE*
23. *TIMESTAMP*
24. *UNIQUEIDENTIFIER*
25. *NVARCHAR*
26. *NCHAR*
27. *VARCHAR*
28. *CHAR*
29. *VARBINARY*
30. *BINARY (*cea mai joasă*)*

- Colaţionarea rezultatului este determinată de regulile de precedenţă a colaţionării atunci când tipul de date este *CHAR, VARCHAR, TEXT, NCHAR, NVARCHAR* sau *NTEXT*.
- Precizia, scara şi lungimea rezultatului depinde de precizia, scara şi lungimea expresiilor de intrare.

3.2. Constrângeri de integritate

Pentru a face ca intrările de date să corespundă maximal celor dezirabile, sistemul SQL Server utilizează şi alte tipuri de constrângeri de integritate, cum sunt cheile primare, valorile implicite pentru câmpuri etc. Dacă integritatea bazei de date nu este asigurată, adică dacă nu sunt respectate constrângerile de integritate, se spune că baza de date este inconsistentă.

Constrângerile au prioritate faţă de declanşatoare (*TRIGGERS*), reguli şi valori implicite (*DEFAULT*). Există cinci tipuri de constrângeri de integritate susţinute de SGBD-ul SQL Server - *NOT NULL*, *UNIQUE*, *PRIMARY KEY*, *FOREIGN KEY* şi *CHECK:*

`NOT NULL`

Specifică faptul că un câmp nu poate avea valori necunoscute, *NULL*.

`UNIQUE`

Determină comportamentul câmpurilor care nu fac parte din cheia primară. Constrângerea *UNIQUE* asigură faptul că valorile duplicate nu vor fi introduse pentru câmpurile specificate. Cu toate acestea, câmpurile specificate pot accepta valori nule, iar sistemul creează automat din ele un index pentru facilitarea căutării în tabele.

`PRIMARY KEY`

Cheie primară reprezintă un câmp sau o mulţime de câmpuri care identifică, în mod univoc, înregistrările unui tabel. Cu alte cuvinte, câmpul, care este cheie primară, nu poate să conţină valori duplicate

Un tabel poate avea doar o singură cheie primară, iar câmpurile care au fost selectate în calitate de câmpuri ale cheii primare nu pot conţine valori necunoscute. Celelalte chei, pe care le poate avea un tabel, se numesc secundare şi ele se folosesc pentru facilitarea accesului la date.

FOREIGN KEY

Cheie externă este un câmp sau o mulțime de câmpuri ce corespund cheii primare a altui tabel Cheia primară se utilizează pentru a defini legăturile dintre tabele.

De exemplu, fie că un tabel, care conține date despre cumpărături, poate avea în calitate de câmp codul furnizorului de la care a fost cumpărată marfa. Câmpul ***Cod_furnizor***, unul din câmpurile altui tabel, care conține date despre vânzători, va fi cheie primară. Evident că ***Cod_furnizor***, din primul tabel, va fi cheie externă pentru tabelul al doilea. Trebuie menționat că, la inserarea unui nou înregistrări în tabelul de cumpărături, valoarea câmpului, care este cheie externă, trebuie să existe în tabelul al doilea, cu care acesta este legat (prin relația **m:1**).

CHECK

Limitează valorile care pot fi luate de un câmp (sau mai multe câmpuri). În cazul actualizării baze de date, sistemul testează constrângerea booleană care urmează după cuvântul rezervat *CHECK*. Dacă constrângerea redată de expresia logică ia valoarea *adevărat*, actualizarea este acceptată. De exemplu, se poate defini o constrângere a domeniului de valori ale câmpului *Salariu,* stabilind salariul funcționarilor între valorile 136 şi 10000:

Salariu >= 136,00 AND Salariu <=10000

3.3. Crearea tabelelor

3.3.1. Un exemplu cu patru tabele

În calitate de primul exemplu, vor fi create patru tabele:

Tabelul ***produse*** conține date despre producător (***Producator***), numărul modelului (***Model***) şi tipul (***Tip***) produsului (*pc, laptop, imprimante*). Se presupune că numărul modelului este unic pentru fiecare producător şi tip de produs. Câmpul ***Model*** este cheie primară pentru tabelul dat.

În ceea ce priveşte tabelul ***pc_uri***, pentru fiecare număr de model, este indicată viteza procesorului, în megahertzi (***Viteza***), capacitatea memoriei ***Ram*** (în Mb), capacitatea discului ***Hd*** (în Gb), viteza dispozitivului de citire ***Cd*** (de exemplu, 4x) şi prețul – ***Pret***. Câmpul ***Cod*** este cheie primară pentru tabelul dat.

Tabelul ***laptop_uri*** este asemănător tabelului ***pc_uri*** cu excepția că, în locul vitezei ***Cd***-ului, este prezentă dimensiunea ecranului – ***Ecran*** (în inch). Câmpul ***Cod*** este cheie primară pentru tabelul dat.

În tabelul ***imprimante***, pentru fiecare model, este indicat faptul dacă ea este color sau nu – ***Color*** ('*da*' - dacă color) şi, de asemenea, tipul imprimantei – ***Tip*** (*Laser, Jet* sau *Matrix*), şi prețul. Câmpul ***Cod*** este cheie primară pentru tabelul dat.

Structura tabelului ***produse*** este următoarea:

Câmp	Tip	Lungimea	Null	Implicit	Observație
Producator	CHAR	1	nu		
Model	VARCHAR	4	nu		
Tip	VARCHAR	10	nu		

Urmează structura tabelului ***pc_uri***:

Câmp	Tip	Lungimea	Null	Implicit	Observație
Cod	INT		nu		
Model	VARCHAR	4	nu		
Viteza	DECIMAL	3, 0	nu		
Ram	DECIMAL	3, 0	nu		
Hd	DECIMAL	2, 0	nu		
Cd	VARCHAR	3	nu		
Pret	FLOAT		nu		

Să se observe structura tabelului ***laptop_uri***:

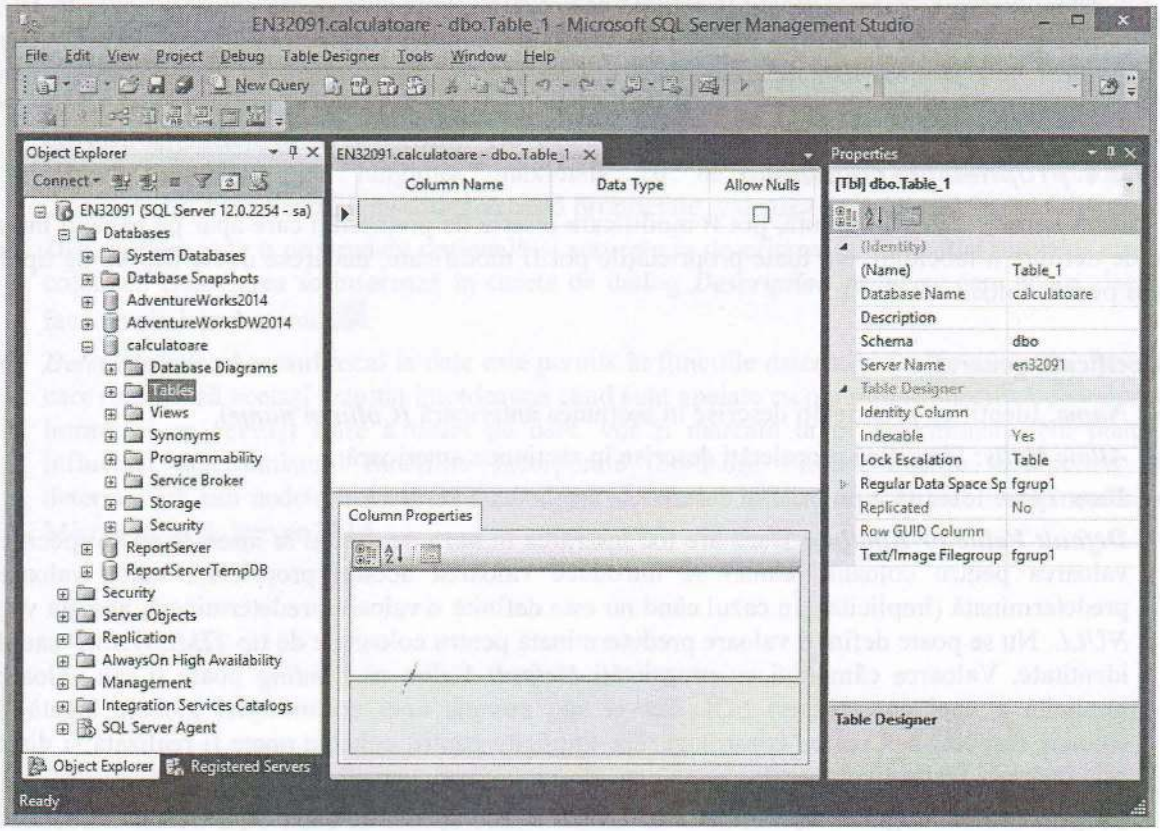
Câmp	Tip	Lungimea	Null	Implicit	Observație
Cod	INT		nu		
Model	VARCHAR	4	nu		
Viteza	DECIMAL	3, 0	nu		
Ram	DECIMAL	3, 0	nu		
Hd	DECIMAL	2, 0	nu		
Pret	FLOAT		nu		
Ecran	INT		nu		

Structura tabelului ***imprimante*** este următoarea:

Câmp	Tip	Lungimea	Null	Implicit	Observație
Cod	INT		nu		
Model	VARCHAR	4	nu		
Color	CHAR	2	nu		
Tip	VARCHAR	6	nu		
Pret	FLOAT		nu		

Acum se va trece la crearea tabelelor. Pentru aceasta, se procedează în felul următor:

1. În arborele de baze de date al ***SQL Server Management Studio***, se face un click pe semnul + al dosarului ***Databases***.
2. Se selectează baza de date în care se vor crea tabelele și se face un click pe semnul + asupra bazei de date alese. Se face un click cu butonul drept al mouse-ului asupra dosarului ***Tables***.
3. În meniul de context, care apare, se selectează opțiunea ***New Table...***,
4. După selectarea opțiunii ***New Table...***, în aria de lucru va apărea următoarea fereastră (filă) a constructorului de tabele:



3.3.2. Definirea proprietăților tabelului

După cum se vede din imaginea de mai sus, în afară de fila constructorului de tabele, în partea dreaptă, se află fereastra ***Properties***, unde pot fi definite proprietățile tabelului. Mai jos, sunt specificate proprietățile care sunt editabile la nivel de tabel.

- ***Name:*** Se specifică numele tabelului.
- ***Description:*** Se adaugă descrierea tabelului.
- ***Schema:*** Se alege schema din care face parte tabelul.
- ***Lock escalation:*** Se setează modul de tratare a blocărilor.
- ***Regular Data Space Specification:*** Se specifică grupurile de fişiere din baza de date sau partițiile din care face parte tabelul.
- ***Text\Image Filegroup:*** Se specifică grupurile de obiecte BLOB din care face parte tabelul.

3.3.3. Definirea coloanelor

Fila de proiectare a tabelului este divizată în două părți: panelul superior, în care se definesc coloanele tabelului, şi panelul inferior, în care se definesc proprietățile fiecărei coloane.

Pentru fiecare coloană a tabelului, se cere completarea unei linii de pe panelul superior al filei de proiectare a tabelului. Pentru fiecare coloană, se cere specificarea, cel puțin, a următoarelor caracteristici:

- ***Column name***: În acest câmp, se scrie numele coloanei (denumirea atributului). Numele este format din litere, cifre, semnul sublinierea (_), semnul aron (@) sau diez (#), dar trebuie să înceapă cu o literă (se permite, de asemenea, şi aplicarea altor simboluri, dar utilizarea lor este recomandată pentru cazuri specifice).
- ***Data Type***: Tipul de date, pe care le poate păstra coloana, sunt cele permise de SGBD-ul SQL Server (vezi tabelele din compartimentul respectiv).
- ***Allow Nulls***: O coloană poate fi definită să accepte sau să nu accepte valoarea specială *NULL*. Se marchează cu o apostilă pătratul respectiv din câmpul ***Allow Nulls***, dacă coloana acceptă valori nule. Sau se demarchează, dacă utilizatorul trebuie neapărat să insereze valori pentru această coloană.

3.3.4. Proprietățile coloanelor

Pentru fiecare coloană definită, pot fi modificate o serie de proprietăți care apar pe panelul inferior al filei de definire a tabelului. Nu toate proprietățile pot fi modificate, deoarece unele depind de tipul de date ales pentru coloana dată.

Specificații generale:

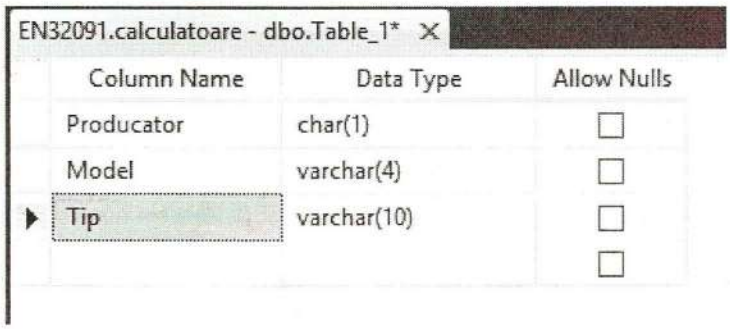
- ***Name***: Identifică proprietăți descrise în secțiunea anterioară ***(Column name).***
- ***Allow Nulls:*** Identifică proprietăți descrise în secțiunea anterioară.
- ***Data Type:*** Identifică proprietăți descrise în secțiunea anterioară.
- ***Default Value or Binding:*** Dacă are loc inserarea în baza de date şi la inserare nu se specifică valoarea pentru coloană, atunci se introduce valoarea acestei proprietăți, adică valoarea predeterminată (implicită). În cazul când nu este definită o valoare predeterminată, aceasta va fi *NULL*. Nu se poate defini o valoare predeterminată pentru coloanele de tip *TIMESTAMP* sau de identitate. Valoarea câmpului cu proprietăți ***Default Value*** or ***Binding*** poate fi sau valoarea implicită a unei constrângeri SQL Server sau numele unei constrângeri globale legată de coloana respectivă. Crearea constrângerilor implicite pentru coloane poate fi realizată şi direct prin tipărirea respectivului text.
- ***Length:*** Indică lungimea tipului de date. La selectarea tipului de date, SQL Server introduce o lungime predefinită pentru acest tip. Această lungime poate fi modificată numai pentru tipurile de date *BINARY*, *CHAR*, *NCHAR*, *VARBINARY*, *VARCHAR* sau *NVARCHAR* (consultați tabelele respective pentru a vedea semnificația concretă a lungimii fiecărui tip de date).

- ***DTS-published***: Arată dacă coloana este editată de serviciul de transformare a datelor (se aplică la bazele de date create, folosind sistemul ***Microsoft SQL Server 2005***).
- ***Full-text Specification:*** Afişează informaţia despre indexul de tip Full-Text. Valoarea acestei proprietăţi este valoarea proprietăţii indexului Full-Text şi indică dacă coloana este sau nu indexată în mod Full-Text. Astfel, pentru modificarea valorii proprietăţii ***Full-Text Specification***, acesta se expandează şi se editează proprietatea respectivă.
  - ***Full-text Indexed:*** Indică dacă această coloană este sau nu indexată full-text. Poate fi setată cu valoarea ***Yes***, doar dacă tipul de date al acestei coloane poate fi explorat în mod full-text şi tabelul ce conţine această coloană posedă un index full-text specificat pentru coloana dată. Pentru editarea proprietăţii respective, se aplică un click pe valoarea ei, se deschide lista drop-down şi se selectează valoarea.
  - ***Full-text Column:*** Afişează numele coloanei care este indexată full-text. Această proprietate trebuie să fie setată, dacă proprietatea ***Datatype*** pentru această coloană are valoarea sau ***image,*** sau ***varbinary***. Coloana căreia i se atribuie această proprietate trebuie să fie de tip ***[n]char, [n]varchar,*** sau ***xml*** şi lista drop-down a acestei proprietăţi conţine doar acele coloane care au unul din aceste 3 tipuri de date.
  - ***Language:*** Indică delimitatorul de cuvinte folosit pentru indexarea unei coloane. Valoarea stocată în proprietatea ***Language*** este, de fapt, însuşi identificatorul local al delimitatorului de cuvinte. Pentru mai multe detalii, să se vadă ***Word Breakers and Stemmers***.
- ***Statistical Semantics:*** Se selectează în cazul în care se doreşte activarea indexarea semanticii statistice pentru coloana selectată.

  Dacă se selectează o limbă, înainte de selectarea ***Statistical Semantics*** şi limba selectată nu are un model asociat de limbă semantică, atunci opţiunea ***Statistical Semantics*** se setează automat la ***No*** şi nu poate fi modificată. Dacă este selectată valoarea ***Yes*** pentru ***Statistical Semantics*** înainte de selectarea unei limbi, atunci limbile disponibile în coloana ***Language*** vor fi restricţionate la cele pentru care există suportul unui model de limbă semantică.
- ***Has Non SQL-Server Subscriber:*** Indică dacă coloana poate fi replicată pe un *subscriber* ce nu e de tip ***SQL Server***. (De exemplu, surse de date tip Oracle, IBM DB2 etc.).
- ***Identity Specifcation:*** Indică faptul că câmpul este autonumeric. Adică, ***SQL Server*** atribuie, în mod automat, o valoare acestei coloane de fiecare dată când se produce inserarea. Coloana nu poate conţine valori nule şi, în mod automat, opţiunea ***Allow Nulls*** va fi demarcată. Această proprietate este disponibilă numai pentru tipurile numerice întregi, de exemplu, *INT*.
  - ***Is Identity***: Indică dacă coloana este sau nu o coloană autonumerică.
  - ***Identity Seed***: Reprezintă valoarea iniţială a unei coloane cu proprietatea ***Identy***.
  - ***Identity Increment:*** Reprezintă creşterea valorii coloanei cu proprietatea ***Identy*** după fiecare inserare.
- ***Indexable:*** Indică dacă câmpul respectiv este indexabil sau nu.
- ***Is Sparse:*** Coloanele *Sparse* sunt coloanele obişnuite care sunt optimizate pentru depozitarea valorilor *NULL*. Coloanele *Sparse* reduc cerinţele de spaţiu pentru valorile necunoscute (*NULL*) în favoarea valorilor cunoscute.
- ***Is Columnset:*** Este un tip de coloană XML, modificabilă şi calculată, care îmbină un set de coloane relaţionale (marcate) într-o singură reprezentare XML.
- ***Merge-published:*** Indică dacă coloana poate fi sau nu publicată prin fuziune. (se aplică la bazele de date create, folosind sistemul ***Microsoft SQL Server 2005***).
- ***Not For Replication:*** Indică dacă valoarea iniţială de identificare este păstrată pe parcursul replicării.
- ***Replicated:*** Identifică câmpuri ce suportă replicarea (se aplică la bazele de date create, folosind sistemul ***Microsoft SQL Server 2005***).
- ***RowGrid:*** Indică faptul că această coloană identifică liniile în context global (nu local, cum o face cheia primară). Se poate aplica numai asupra coloanei de tip *UNIQUEIDENTIFIER* şi, în tabel, nu poate exista mai mult de o coloană de acest tip.

- ***Size:*** Arată dimensiunea în octeți permisă de tipul de date al coloanelor. De exemplu, tipul de date *NCHAR* poate avea lungimea 10 (numărul de caractere), dar acesta va avea lungimea 20 pentru setul de caractere *UNICODE*.

Folosind cele prezentate până aici, se poate trece la construirea structurii tabelului (schemei relației) ***produse***. La sfârşitul procesului, structura poate arăta precum cea din fereastra ce urmează:

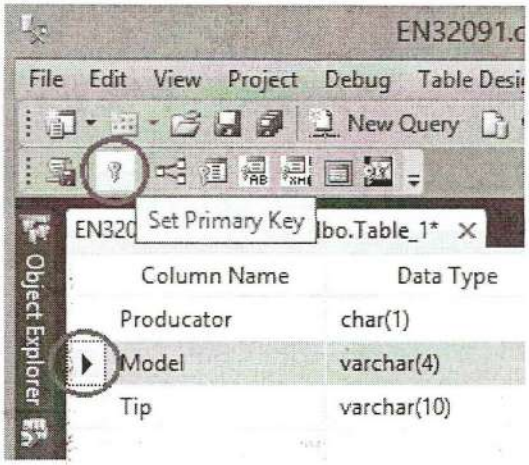


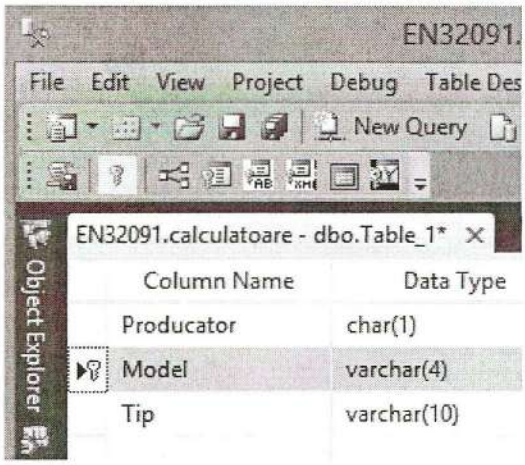
3.3.5. Stabilirea cheii primare

Pentru stabilirea cheii primare, este suficientă selectarea (în panelul superior al filei de proiectare a tabelelor) coloanei anterioare celei ***Column Name*** şi apoi aplicarea unui click pe butonul **Set Primary Key**, , de pe bara de instrumente. În stânga coloanei selectate, apare o iconiță în formă de cheiță, simbolizând că această coloană este cheia primară a tabelului.

În cazul când cheia primară este compusă (are mai mult de o coloană), procesul urmat este acelaşi ca şi pentru cheia primară simplă. Aici, pentru selectarea mai multor coloane, atunci când se aplică clickurile pe câmpurile din stânga coloanei ***Column Name,*** se ține apăsată tasta **Ctrl**.

În cazul nostru, câmpul ***Model*** este cheie primară pentru tabelul ***produse***. În prima figură, ce urmează, sunt încercuite cu roşu coloana ce precede coloana ***Column Name*** (pe care se face un click) şi butonul **Set Primary Key**. După finalizarea acestui proces, câmpul ***Model*** va rămâne cu cheia poziționată în partea sa stângă, precum e arătat în figura a doua ce urmează:





3.3.6. Salvarea tabelului

Îndată ce tabelul este complet definit, el trebuie salvat. Pentru aceasta, este suficientă efectuarea unui click pe butonul **Save** , , de pe bara de instrumente. Dacă numele tabelului nu a fost introdus în fereastra ***Properties*** (secţiunea 3.3.1 a acestui capitol), în acest moment, va apărea fereastra ***Choose Name***, unde va trebui inserat şi apoi confirmat numele tabelului, aplicând un click pe butonul **OK**.

3.4. Ştergerea, redenumirea şi actualizarea tabelelor

3.4.1. Ştergerea tabelelor din baza de date

Câteodată, este necesară ştergerea unui tabel (de exemplu, dacă se doreşte implementarea unui nou proiect al bazei de date sau dacă se doreşte eliberarea spaţiului ocupat de baza de date). Trebuie să se ţină cont că, atunci când se şterge un tabel, definiţia sa structurală, datele, indecşii full-text, constrângerile şi indecşii sunt şterşi pentru totdeauna din baza de date, iar spaţiul care era utilizat pentru stocarea tabelului şi indecşilor lui este oferit altor tabele.

Dacă se doreşte ştergerea tabelelor care sunt legate cu constrângeri de tipul ***Foreign Key***, ***Unique*** sau ***Primary Key***, trebuie să se şteargă, mai întâi, tabelele cu constrângeri ***Foreign Key***. Dacă se doreşte ştergerea tabelului care este referit într-o constrângere ***Foreign Key***, dar nu se doreşte ştergerea întregului tabel cu cheie străină, trebuie să se şteargă constrângerea ***Foreign Key***.

Pentru ştergerea completă a unui tabel din baza de date, se procedează în modul următor:

1. În baza de date unde a fost creat tabelul, se selectează (cu un click) dosarul ***Tables***. Se observă că tabelul căutat va apărea în partea dreaptă, în lista de obiecte de pe fila ***Object Explorer Details***.
2. Se face un click cu butonul drept al mouse-ului pe tabelul care trebuie suprimat.
3. În meniul de context care se deschide, se selectează opţiunea ***Delete***. În consecinţă, pe ecranul calculatorului, va fi expusă o fereastră care oferă o scurtă informaţie despre obiectul preconizat pentru ştergere (numele obiectului, tipul, proprietarul, statutul, mesajele). Tot în această fereastră, apăsând butonul ***Show Dependencies,*** pot fi vizualizate dependenţele logice ale obiectului dat cu alte obiecte şi structuri din schema globală a bazei de date.
4. Se dă un click pe butonul **OK** al ferestrei ***Delete Object***, după care tabelul respectiv va fi şters din baza de date.

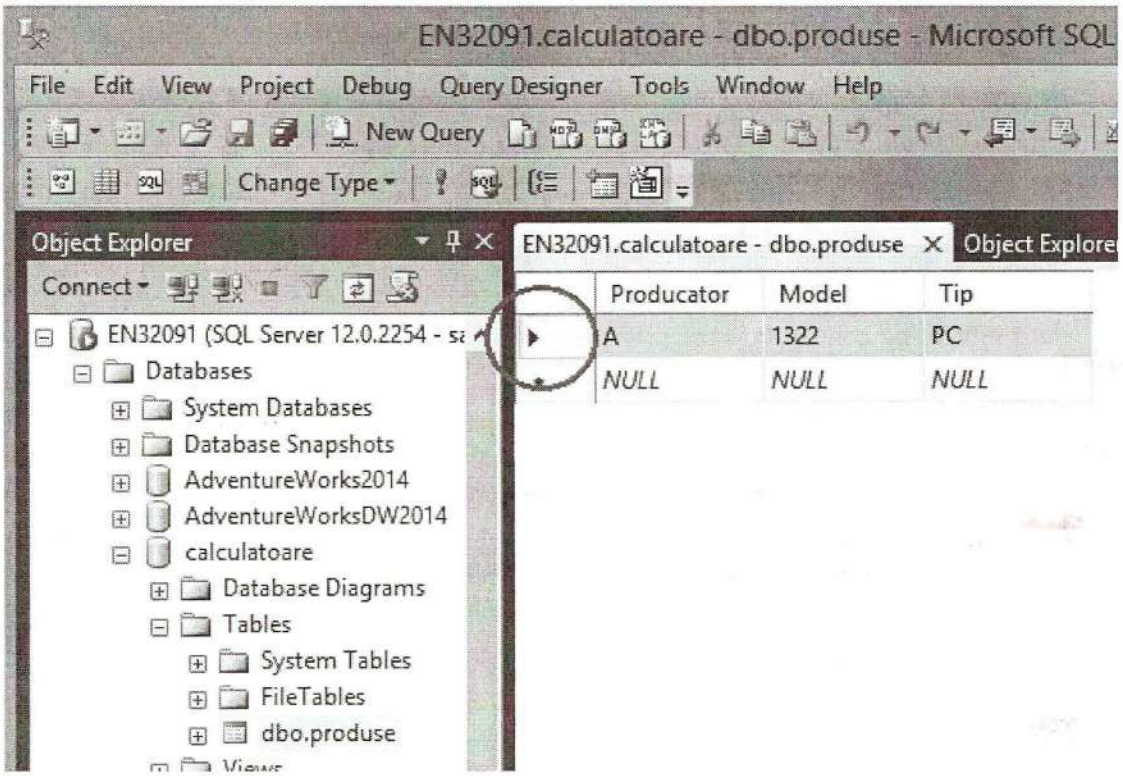
3.4.2. Redenumirea tabelelor

Procesul de redenumire a tabelelor, practic, este asemănător celui anterior. Pentru a redenumi un tabel, în meniul de context, se selectează opţiunea ***Rename***, în loc de opţiunea ***Delete***, după care se inserează numele nou al tabelului.

3.4.3. Introducerea, editarea şi ştergerea datelor din tabele

Introducerea, editarea şi ştergerea datelor din tabele sunt operaţii foarte simplu de realizat. Pentru a avea accesul la aceste operaţii, se face un click cu butonul drept al mouse-ului asupra tabelului în cauză şi, în meniul de context care apare, se selectează opţiunea ***Edit Top 200 Rows.***

După selectarea respectivei opţiuni, pe ecranul calculatorului apare tabelul cu conţinutul lui:



În această fereastră, pot fi introduse, modificate sau şterse înregistrări:

- Pentru introducerea înregistrărilor, se aplică un click în câmpul liber al coloanei ***Producator*** şi se iniţiază scrierea datelor în înregistrarea nouă. Ajungând la sfârşitul acestei înregistrări, la coloana ***Activ***, după includerea valorii lui, automat este deschisă o înregistrare nouă.
- Pentru modificarea datelor unui câmp al unei înregistrări, este suficient să se aplice un click asupra acestui câmp, după care se poate modifica conţinutul lui.
- Pentru excluderea unei înregistrări, se face un click drept pe pointerul acesteia (încercuit cu roşu) şi în meniul de context se alege ***Delete***. Pe ecran, apare o fereastră de avertizare în care se cere confirmarea operaţiei de ştergere.

3.5. Modificarea structurii unui tabel

Se poate întâmpla că, după proiectarea structurii unui tabel, să apară necesitatea includerii sau excluderii unor câmpuri, deja, create. Aceste sarcini sunt simple în cazul când tabelul în cauză nu este completat cu date.

Cu toate acestea, SQL Server permite să se adauge tabelelor (cu date) existente coloane noi, dacă aceste coloane acceptă valori nule sau sunt create constrângeri ***Default*** pe aceste coloane. Atunci când într-un tabel se adaugă o coloană nouă, SQL Server inserează o valoare în coloană pe fiecare înregistrare existentă. De aceea, atunci când coloana este adăugată unui tabel, este foarte bine să se specifice pentru coloană o constrângere ***Default***. Dacă coloana nouă nu are o definiţie ***Default***, trebuie să se specifice că coloana acceptă valori nule. SQL Server va insera, în acest caz, în coloană, valori *NULL* şi va returna o eroare, dacă coloana nouă nu acceptă valori *NULL*.

3.5.1. Modificarea proprietăţilor coloanelor

Coloanele pot fi şterse din tabelele existente sau pot fi modificate proprietăţile coloanelor. Însă, există cazuri speciale când proprietăţile coloanei nu pot fi modificate:

- Coloana este de tipul *TIMESTAMP* sau *UNIQUEIDENTIFIER*.
- Este o coloană calculată (*computed*) sau este utilizată de o coloană calculată.
- Este folosită într-un index. Excepţie pot fi coloanele de tip *VARCHAR, NVARCHAR* sau *VARBINARY,* dacă tipul de date nu se schimbă, ci se măreşte dimensiunea şi indexul dat nu este rezultatul constrângerii *PRIMARY KEY*.

- Coloana este utilizată de statisticile generate de instrucţiunea *CREATE STATISTICS*. Excepţie pot fi coloanele de tip *VARCHAR, NVARCHAR* sau *VARBINARY,* dacă tipul de date nu se schimbă, ci se măreşte dimensiunea acestuia sau coloana se schimbă din *NOT NULL* în *NULL*.
- Coloana este utilizată în *PRIMARY KEY* sau în constrângerea [*FOREIGN KEY*] *REFERENCES*.
- Coloana este utilizată în constrângerile *CHECK* sau *UNIQUE*. Însă schimbarea lungimii coloanelor utilizate în *CHECK* sau *UNIQUE* este posibilă.
- Coloanele asociate cu clauza *DEFAULT*. Dar, totuşi, este posibilă modificarea lungimii, preciziei sau scalei coloanei, dacă tipul de date nu este modificat.
- Dacă tabelul ce conţine coloana este partiţionat, atunci schimbarea tipului de date al acestei coloane nu este posibilă.

Există şi unele constrângeri în ce priveşte modificarea coloanelor de tipul *TEXT*, *NTEXT* sau *IMAGE*. Atunci când se doreşte o modificare a tipurilor de date, trebuie să se ţină cont de faptul că aceste modificări pot duce la modificarea datelor. De exemplu, reducerea preciziei unei coloane poate duce la trunchierea datelor conţinute.

Dacă constrângerile sus-numite sunt respectate şi tabelul nu conţine date care pot fi pierdute în urma operaţiilor de modificare sau ştergere a coloanelor, atunci modificarea, inserarea sau ştergerea coloanelor se realizează, precum urmează.

3.5.2. Inserarea unui câmp nou

Pentru inserarea unui câmp în structura unui tabel:

1. Se face un click cu butonul drept al mouse-ului asupra tabelului care trebuie să posede un câmp nou.
2. În meniul de context, care s-a deschis, se acţionează opţiunea ***Design.***
3. Se selectează coloana (câmpul), înainte căreia trebuie să fie plasată coloana nouă, executând un click cu butonul drept al mouse-ului.
4. În meniul de context, care s-a deschis, se selectează opţiunea ***Insert Column.***
5. Se completează proprietăţile pentru coloana nouă.
6. Se face un click pe butonul care reprezintă acţiunea de salvare (**Save**).

   Tot aici pot fi editate coloanele deja existente, selectând cu un clic setările ce trebuie supuse schimbării.

3.5.3. Ştergerea unui câmp existent

Pentru ştergerea unui câmp din structura unui tabel:

1. Se execută primii 2 paşi din procedura anterioară (de inserare).
2. Se face un click cu butonul drept al mouse-ului pe coloana care trebuie să fie exclusă.
3. În meniul de context care s-a deschis, se acţionează opţiunea ***Delete Column.***
4. Se execută un click pe butonul **Save** de pe bara de instrumente.

Întrebări de control:

1. Ce proprietăţi obligatorii trebuie să posede o coloană?
2. Care sunt tipurile de date utilizate de sistemul SQL Server?
3. Ce constrângeri de integritate utilizează SQL Server pentru menţinerea consistenţei bazei de date?
4. Ce dificultăţi şi urmări pot apărea în procesul de ştergere a câmpurilor din tabelele, deja, existente?
5. Ce dificultăţi şi urmări pot apărea în procesul de modificare a câmpurilor din tabelele, deja, existente? Care câmpuri nu pot fi modificate?

Sarcini practice:

1. Care din numerele prezentate mai jos pot fi introduse într-un câmp de tipul *DECIMAL(4,1)*?

   a) 16,2; **b)** 116,2; **c)** 16,21; **d)** 1116,2; **e)** 1116,21.

2. Creaţi o bază de date numită ***calculatoare*** cu proprietăţi implicite. Creaţi, în cadrul acestei baze de date, cele patru tabele (***produse***, ***pc_uri***, ***laptop_uri*** şi ***imprimante***), schemele cărora sunt definite în secţiunea 3.3 al capitolului.
3. Inseraţi în tabelele respective ale bazei de date ***calculatoare*** următoarele înregistrări:

produse	Producator	Model	Tip
	B	1121	PC
	A	1232	PC
	A	1233	PC
	E	1260	PC
	A	1276	Imprimante
	D	1288	Imprimante
	A	1298	Laptop_uri
	C	1321	Laptop_uri
	A	1401	Imprimante
	A	1408	Imprimante
	D	1433	Imprimante
	E	1434	Imprimante
	B	1750	Laptop_uri
	A	1752	Laptop_uri
	E	2111	PC
	E	2112	PC

pc_uri	Cod	Model	Viteza	Ram	Hd	Cd	Pret
	1	1232	500	64	5	12x	600
	2	1121	750	128	14	40x	850
	3	1233	500	64	5	12x	600
	4	1121	600	128	14	40x	850
	5	1121	600	128	8	40x	850
	6	1233	750	128	20	50x	950
	7	1232	500	32	10	12x	400
	8	1232	450	64	8	24x	350
	9	1232	450	32	10	24x	350
	10	1260	500	32	10	12x	350
	11	1233	900	128	40	40x	980

laptop_uri	Cod	Model	Viteza	Ram	Hd	Pret	Ecran
	1	1298	350	32	4	700	11
	2	1321	500	64	8	970	12
	3	1750	750	128	12	1200	14
	4	1298	600	64	10	1050	15
	5	1752	750	128	10	1150	14
	6	1298	450	64	10	950	12

imprimante	**Cod**	**Model**	**Color**	**Tip**	**Pret**
	1	1276	nu	Laser	400
	2	1433	da	Jet	270
	3	1434	da	Jet	290
	4	1401	nu	Matrix	150
	5	1408	nu	Matrix	270
	6	1288	nu	Laser	400

4. INSTRUCȚIUNEA SELECT TRANSACT-SQL

Capitolul este consacrat studierii elementelor de bază ale limbajului Transact-SQL, a structurii instrucțiunii *SELECT* şi a tipurilor de funcții utilizate în interogări. În afară de aceasta, se tratează aspectele de utilizare a interfeței *Editorului de Interogări* pentru crearea şi gestiunea interogărilor SQL [2].

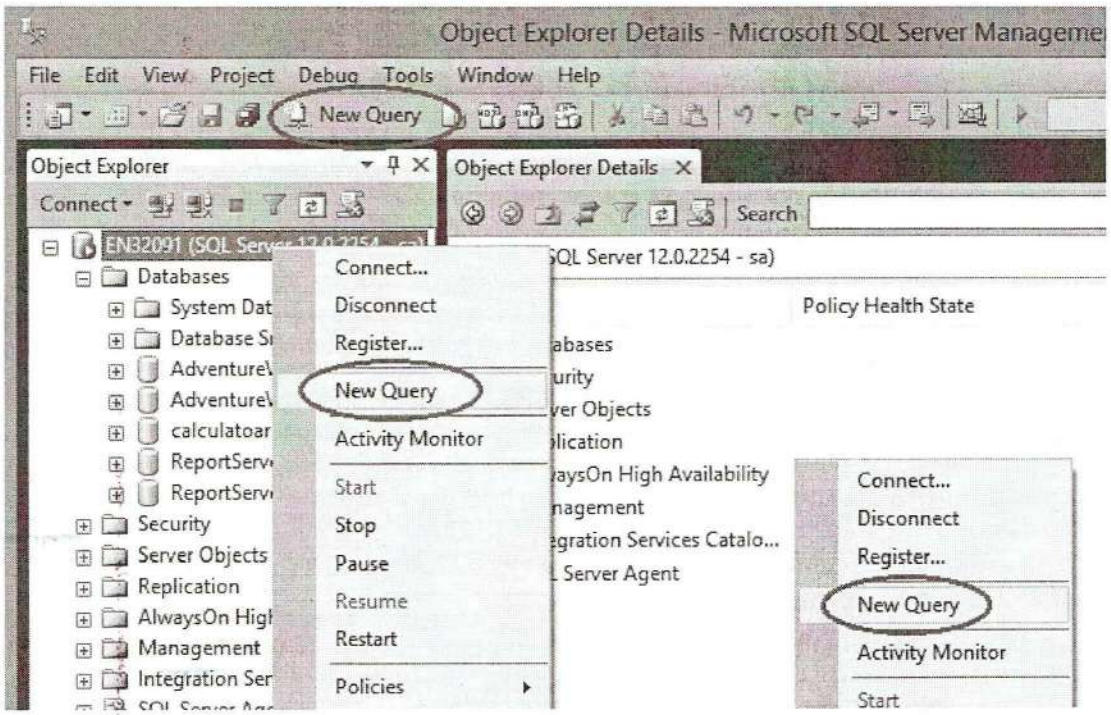
Înainte de a trece la studierea limbajului SQL, se va examina sumar instrumentele grafice de construire a interogărilor oferite de editorul de interogări al sistemului SQL Server. Aceste instrumente asigură o serie de facilități legate de recuperarea şi optimizarea interogărilor. Între ele, pot fi citate:

- Vizualizarea datelor sub formă de text sau grid, utilizând, pentru aceasta, comenzile SQL.
- Analiza planului de executare a interogării.
- Helpul on-line al limbajului Transact-SQL.

4.1. Încărcarea Transact-SQL Query Editor

Pentru încărcarea Transact-SQL Query Editor, se execută următoarele operații:

1. Se lansează interfața SQL Server Management Studio, se alege serverul respectiv din lista de servere înregistrate.
2. În continuare, se face un click pe primul buton de pe bara de instrumente, ***New Query***, precum este prezentat în figura de mai jos (varianta 1). O altă cale este aplicarea unui click cu butonul drept al mouse-ului asupra serverului respectiv şi din meniul din context apărut se alege opțiunea ***New Query*** (varianta 2). Aceeaşi opțiune poate fi selectată, aplicând un click-drept în câmpul de lucru *Management Studio* (varianta 3).



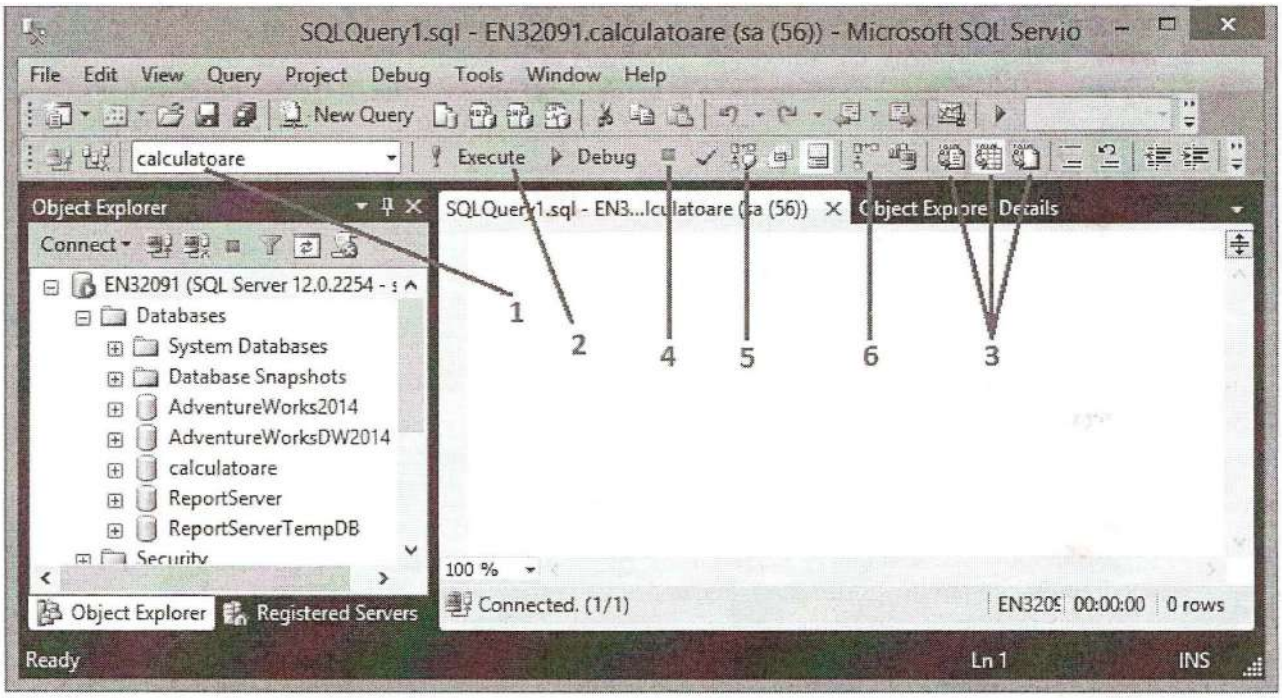
După ce se apasă ***New Query*** apare pagina în care se va lucra. Spațiul de lucru (în dreapta) este destinat scrierii instrucțiunilor SQL. În plus, există o fereastră paralelă cu prima, în care se afişează rezultatul interogării sau mesajele de erori. Fereastra se deschide automat odată cu executarea interogării.

4.2. Funcționalitățile componentei Transact-SQL Query Editor

Transact-SQL Query Editor reprezintă editorul de interogări scrise în limbajul Transact-SQL.

În primul rând, pe bara de instrumente, în căsuța de dialog ***Available Databases*** (1), poate fi selectată baza de date cu care se va lucra. În cazul de față, se va selecta baza de date-test, numită ***calculatoare***, care a fost creată în lucrările precedente.

Dacă este scrisă o instrucțiune SQL, executarea acestei instrucțiuni de către sistem este solicitată prin aplicarea unui click pe butonul de control (2) din imaginea de mai jos sau prin apăsarea tastei F5.



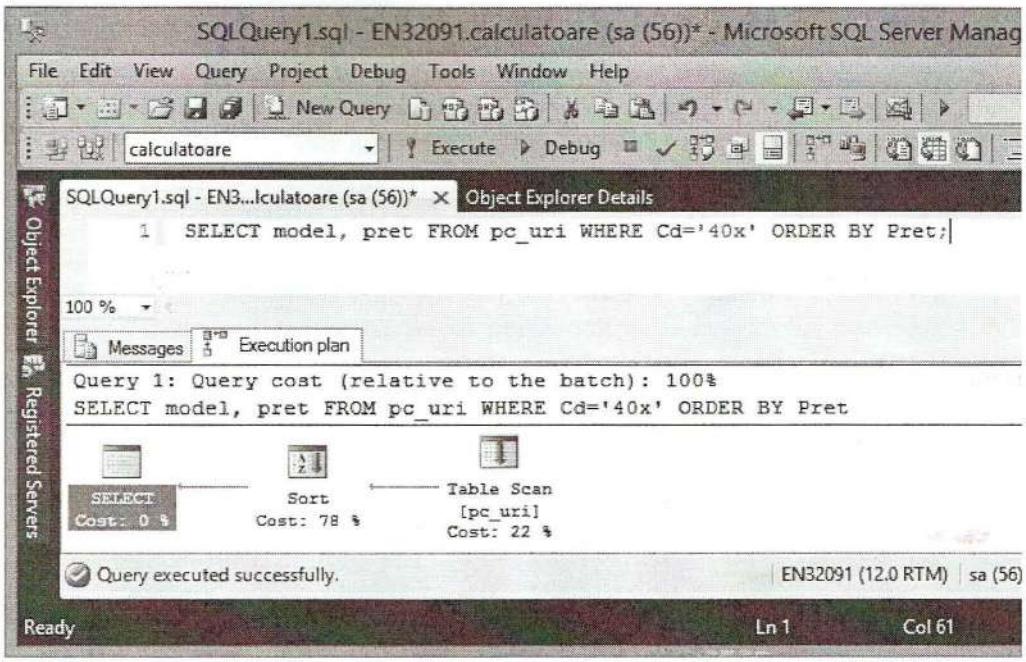
În cazul în care rezultatul se cere să fie afișat sub formă de grid sau sub altă formă, se aplică un click pe butonul respectiv de pe bara de instrumente (3) ***Results to grid*** sau ***text***, sau ***file.*** Unul din cele mai mari avantaje ale utilizării opțiunii grid constă în faptul că coloanele pot fi ajustate la necesitățile practice. Dimensiunea celulelor-grid pot fi modificate asemenea celulelor Excel, de exemplu.

Apăsând butonul (4) ***Cancel Executing Query***, serverului i se trimite o cerere de anulare a interogării. Unele interogări nu pot fi anulate imediat, ci trebuie așteptată o stare potrivită de anulare. Fiind anulată, o retardare apare datorită procesului de desfășurare (*rollback*) a tranzacțiilor.

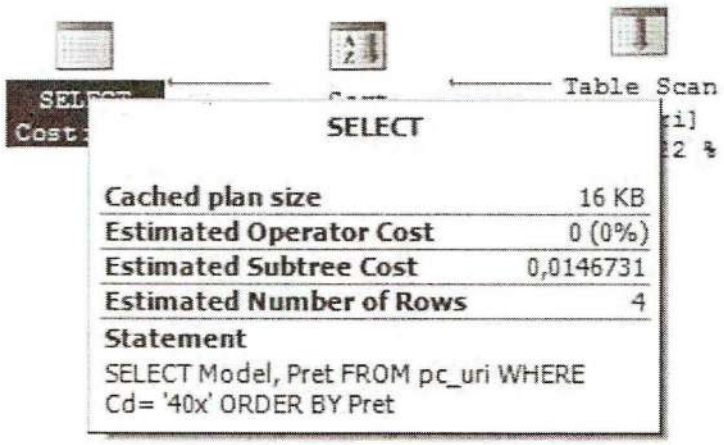
Butonul numerotat în imaginea de mai sus cu cifra (5), numit ***Display Estimated Execution Plan,*** are funcția de a solicita, de la procesorul de interogări, un plan de executare a interogării, fără ca realmente aceasta să fie executată, iar planul să fie afișat în fereastra ***Execution plan***. Acest plan utilizează index-statistici pentru estimarea numărului de rânduri așteptate care vor fi returnate pe parcursul fiecărei părți de executare a interogării. Planul real de executare poate fi diferit de planul estimat de executare, dacă numărul de rânduri returnate se deosebește esențial de cel evaluat și procesorul de interogări a modificat planul pentru a-l face mai eficient

Fie că se scrie o instrucțiune asupra unui tabel din baza de date și fie că s-a aplicat un click pe butonul în chestiune, ***Display Estimated Execution Plan***. Să se observe graficul pe care l-a generat interogarea.

Interogarea cere de la sistem să fie selectate câmpurile ***Model*** (modelul calculatorului) și ***Pret*** (prețul) ale tabelului ***pc_uri*** din înregistrările în care viteza Cd-ului este 40x, și cere ca înregistrările rezultatului să fie prezentate în ordinea (ascendentă) câmpului ***Pret***. După scrierea acestei instrucțiuni, se face un click pe butonul (5), după care se obține rezultatul ce urmează:



În afară de această analiză grafică, pot fi obținute date despre orice etapă a interogării. Cu acest scop, este suficientă poziționarea cursorului mouse-ului asupra oricărei iconițe a prezentării grafice a interogării și imediat este expusă o fereastră asemănătoare celei ce urmează:



Pentru a evalua costul real al interogării executate, se apasă butonul (6), ***Include Actual Execution Plan***. Este executată interogarea, sunt obținute rezultatele interogării și planul de execuție a interogării, în forma grafică în fereastra ***Execution plan***.

Dacă este necesar, interogarea poate fi salvată, apoi recuperată, folosind opțiunile cunoscute ***Save*** și ***Open*** pentru fișiere, care sunt prezente în majoritatea componentelor sistemului Windows.

4.3. Elemente de bază ale instrucțiunii SELECT

Următorul pas îl constituie studierea limbajului de interogări. Întotdeauna, când vor apărea probleme în sintaxa unei instrucțiuni, se poate apela la ajutorul componentei ***Help On Line***. Pentru aceasta, este suficient să se selecteze în fereastră elementul instrucțiunii Transact-SQL și să se acționeze tasta **F1**. În consecință, se va deschide fereastra ***SQL Server Books Online***, unde poate fi vizualizată sintaxa completă a instrucțiunii.

Până aici, activitățile au fost îndreptate spre administrarea mecanismelor bazelor de date, crearea tabelelor etc. Dar niciuna din aceste activități nu are sens, dacă nu se cunoaște limbajul de interogări.

În acest capitol, vor fi studiate instrucțiunile principale ale limbajului, pentru a ne obișnui cu utilizarea limbajului SQL, apoi, pe parcurs, vor fi examinate aspecte mai aprofundate ale acestui subiect.

Mai întâi, trebuie menționat că instrucțiunile SQL se împart în 4 grupuri principale: DDL, DML, DCL și TCL. Care e deosebirea și pentru ce sunt folosite fiecare din ele?

DDL (*data definition language*) este o parte a limbajului SQL, care este utilizată pentru definirea datelor și obiectelor unei baze de date. Atunci când sunt utilizate instrucțiunile din acest grup, în dicționarul de date al sistemului SQL Server sunt realizate intrările respective. Cu ajutorul acestor instrucțiuni pot fi create tabele și (sau) modificate tabelele specificate, inserate coloane (câmpuri) în tabele, definite constrângeri de integritate, construiți indecși etc.

DML (*data manipulation language*) este o parte a limbajului SQL, care este utilizată pentru manipularea datelor, adică consultarea, inserarea liniilor (înregistrări) în tabele, excluderea liniilor și modificarea liniilor.

DCL (*data control language*) este utilizat pentru crearea rolurilor, permisiunilor și integrităților referențiale. De asemenea, este utilizat pentru administrarea accesului la baze de date și la resursele serverului.

TCL (*Transactional Control Language*), este utilizat pentru gestionarea tranzacțiilor care au loc într-o bază de date.

În cadrul acestui volum detaliat vor fi analizate instrucțiunile DML și DDL.

4.3.1. Operatori și expresii utilizate în interogări

Expresiile sunt elemente utilizate pentru definirea criteriilor de selecție, regulilor de validare sau câmpurilor calculate, formate dintr-o combinație de operatori, constante, variabile și funcții ce au ca rezultat o anumită valoare. În construcția instrucțiunilor SQL, se obișnuiește să se folosească diverse expresii și operatori.

Două expresii pot fi combinate cu un operator în cazul în care ambele au tipuri de date acceptate de către operator și îndeplinesc cel puțin una dintre aceste condiții:

- Expresiile au același tip de date.
- Tipul de date cu prioritate mai mică pot fi convertite implicit la tipul de date de prioritate mai mare.
- Funcția *CAST* poate converti, în mod explicit, tipul de date cu prioritate mai mică la tipul de date cu prioritate mai mare sau un tip de date intermediar care poate fi transformat implicit la tipul de date cu cea mai mare prioritate.

Operatorii utilizați în construirea expresiilor Transact-SQL se pot clasifica astfel:

Operatori aritmetici	Reprezentare
Împărțirea	/
Modulul	%
Înmulțirea	*
Adunarea	+
Scăderea	-

Operatori de caracter	Reprezentare
Concatenarea	+

Operatori de comparare	Reprezentare
Diferit de	<>
Egal cu	=
Mai mare sau egal cu	>=
Mai mare ca	>
Mai mic sau egal cu	<=
Mai mic ca	<
Nu e egal cu	!=
Nu e mai mare ca	!>
Nu e mai mic ca	!<

Operatori la nivel de biți	Reprezentare
Operatorul *AND* pe biți	&
Operatorul *NOT* pe biți	~
Operatorul *OR* pe biți	|
Operatorul exclusive *OR* pe biți	^

Operatori logici	Reprezentare
Adevărat, dacă pentru orice element din mulțimea definită de expresie este adevărat	*ALL*
Adevărat, în cazul când operandul este din intervalul fixat	*BETWEEN*
Adevărat, în cazul când o subinterogare extrage cel puțin un rând	*EXISTS*
Adevărat, când un element se găsește într-o mulțime de elemente	*IN*
Adevărat, în cazul când operandul găsește un element ce se potrivește cu un model	*LIKE*
Inversează valoarea booleană	*NOT*
Adevărat, în cazul când vreun element din mulțimea definită de expresie are valoarea adevărat.	*SOME*
Adevărat, în cazul când vreun element din mulțimea definită de expresie are valoarea adevărat.	*ANY*
Adevărat, dacă ambele expresii în structura instrucțiunii au valoarea adevărat	*AND*
Adevărat dacă cel puțin una din expresiile constante în structura comenzilor are valoarea adevărat	*OR*

4.3.2. Interogări simple

Instrucțiunea *SELECT* reprezintă blocul de interogare de bază și ea selectează informațiile dorite din tabelele bazei de date. Sintaxa acestei instrucțiuni este întotdeauna aceeași, este foarte puternică și are următoarea formă generală:

```
SELECT [ALL|DISTINCT] <listă coloane>
       FROM <listă tabele>
       [WHERE <condiție căutare>] [<clauze_secundare>]
```

Astfel, o instrucțiune *SELECT* este formată din mai multe cuvinte-cheie numite clauze. *SELECT* se definește utilizând diverse configurații ale acestor clauze în scopul obținerii informației necesare. Unele clauze sunt obligatorii, altele opționale. În plus, fiecare clauză este formată dintr-unul sau mai multe cuvinte-cheie utilizate la extragerea ca întreg a informației cerute prin enunțul *SELECT*.

Se remarcă trei secțiuni importante ale construcției de interogare: clauza *SELECT*, clauza *FROM* și clauza *WHERE*:

- Clauza `SELECT` este obligatorie. Se utilizează pentru a specifica numele coloanelor (câmpurile) ce vor fi afișate în rezultatul interogării.
- `<listă coloane>` cuprinde câmpurile care trebuie să apară în tabelul sau tabelele cu rezultatele interogării.
- `ALL` cere includerea tuturor înregistrărilor care îndeplinesc condițiile impuse. Instrucțiunile *SELECT* `<listă coloane>`... și *SELECT ALL* `<listă coloane>`... au același rezultat practic, calificativul *ALL* este rar folosit.
- `DISTINCT` cere eliminarea înregistrărilor care conțin duplicate în câmpurile selectate, afișând numai o apariție a acesteia.
- Clauza `FROM` este, de asemenea, obligatorie. *FROM* se utilizează pentru a specifica tabelele din care vor fi extrase coloanele specificate în clauza *SELECT*.
- `<listă tabele>` reprezintă lista tabelelor de unde datele trebuie să fie selectate.
- `WHERE` – clauză opțională utilizată în filtrarea înregistrărilor returnate de clauza *FROM*. *WHERE* este precedat de o expresie numită condiție de căutare.
- `<condiție căutare>` – definește un filtru pentru interogare, folosind clauza *WHERE*.

Clauza *SELECT* introduce lista coloanelor sau al expresiilor care vor fi selectate și afișate. În cazul în care se cere selectarea mai multor câmpuri, acestea se separă prin virgulă.

Să se revină la următoarea instrucțiune (secțiunea 4.2.) și să se tipărească în ***Transact-SQL Query Editor***. Apoi să se execute instrucțiunea în modul acceptat (sub formă de listă sau grid) și apoi să se execute ***Display Estimated Execution Plan***:

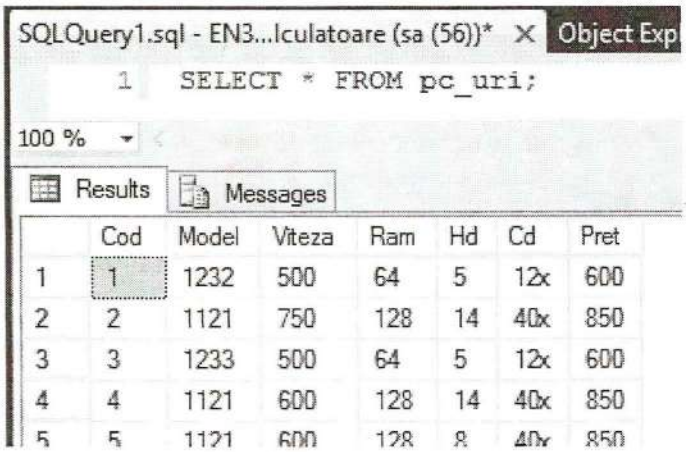
```
SELECT Model, Pret FROM pc_uri WHERE Cd='40x' ORDER BY Pret
```

Să se analizeze rezultatul.

Se examinează un exemplu. În ***Transact-SQL Query Editor***, se tipăreşte următoarea instrucțiune:

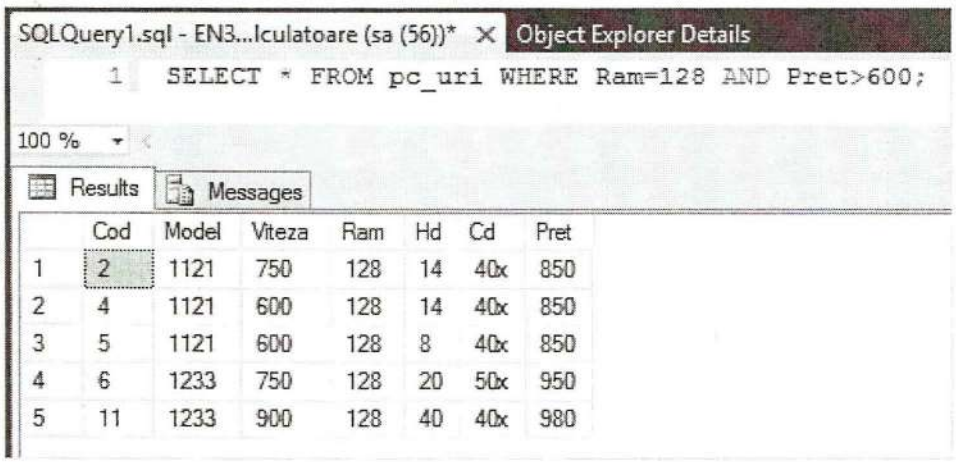
```
SELECT * FROM pc_uri
```

Daca lista de câmpuri este un asterisc (*), atunci se selectează toate câmpurile tabelelor indicate prin clauza *FROM*. Astfel, se cere selectarea tuturor câmpurilor tabelului ***pc_uri***. Rezultatul, după executarea instrucțiunii sub formă de grid, va fi următorul:



	Cod	Model	Viteza	Ram	Hd	Cd	Pret
1	1	1232	500	64	5	12x	600
2	2	1121	750	128	14	40x	850
3	3	1233	500	64	5	12x	600
4	4	1121	600	128	14	40x	850
5	5	1121	600	128	8	40x	850

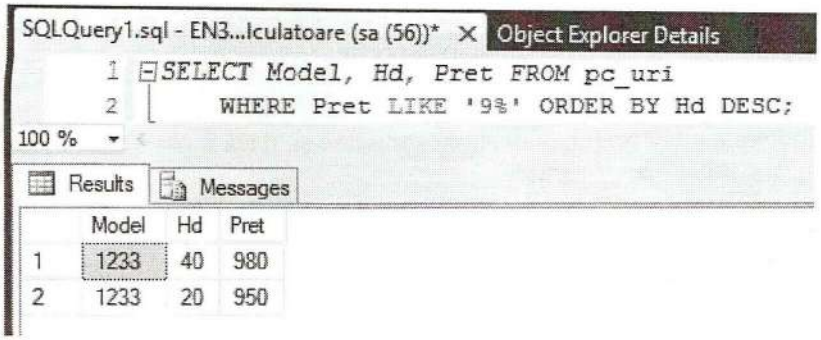
Clauza *WHERE* restricționează liniile returnate ca rezultat la acele linii care îndeplinesc condiția introdusă de această clauză. În forma cea mai obişnuită, clauza *WHERE* este urmată de o condiție, dată ca o expresie booleană. Se pot combina două sau mai multe condiții folosind operatorii logici *OR* şi *AND*. De aceea, e lesne de observat faptul că există o mulțime de combinații care pot fi realizate. Spre exemplu, se examinează o altă instrucțiune:



	Cod	Model	Viteza	Ram	Hd	Cd	Pret
1	2	1121	750	128	14	40x	850
2	4	1121	600	128	14	40x	850
3	5	1121	600	128	8	40x	850
4	6	1233	750	128	20	50x	950
5	11	1233	900	128	40	40x	980

Se selectează toate câmpurile tabelului ***pc_uri,*** pentru care (*WHERE*) capacitatea memoriei operative (***Ram***) este egală cu *128* şi (*AND*) prețul (***Pret***) este mai mare de 600. Astfel, este formulată interogarea care îndeplineşte condiția din clauza *WHERE*.

În următorul exemplu, este formulată o interogare care cere ca înregistrările să fie reprezentate sub formă de grid, cu câmpurile ***Model, Hd, Pret***, unde câmpul ***Pret*** începe cu cifra (simbolul) '9' şi rezultatul să fie ordonat pe câmpul ***Hd*** în ordine descendentă.



După cum se poate observa, clauza *LIKE* testează dacă valoarea parțială a unui șir de caractere coincide cu un șablon definit de utilizator. Această clauza face uz de caracterul procent (%) în calitate de șablon, care denotă o secvență de lungime arbitrară de caractere ce pornește din această poziție.

Clauza *ORDER BY* introduce numele câmpului, după care se face ordonarea liniilor rezultate. Ordonarea este, implicit, în ordine crescătoare. Dacă numele câmpului este urmat de cuvântul *DESC*, ordonarea liniilor se face în ordine descrescătoare a valorilor acelui câmp.

4.3.3. Funcții pe un sigur rând

O funcție în limbajul Transact-SQL nu este altceva decât o rutină care realizează o operație anumită și întoarce un rezultat. Funcțiile constituie o caracteristică importantă a limbajului și sunt utilizate pentru:

- a realiza calcule asupra datelor;
- a modifica date;
- a manipula grupuri de înregistrări;
- a schimba formatul datelor;
- sau pentru a converti diferite tipuri de date.

O funcție este constituită dintr-o serie de argumente, care sunt utilizate în procesare. Sintaxa de bază este următoarea:

```
<nume funcție> (<argument1> [, <argument2> ...] )
```

Funcțiile se clasifică în două tipuri:

- ***Single row*** – returnează rezultatul pentru fiecare linie a unui tabel.
- ***Group*** sau ***multiple row***– returnează un singur rezultat pentru un grup de linii.

Diferența dintre cele două tipuri de funcții este numărul de înregistrări pe care acționează: funcțiile referitoare la o singură înregistrare returnează un singur rezultat pentru fiecare rând al tabelului, pe când funcțiile referitoare la mai multe înregistrări returnează un singur rezultat pentru fiecare grup de înregistrări din tabel.

Funcțiile referitoare la o singură înregistrare sunt funcții utilizate pentru manipularea datelor individuale. Ele pot avea unul sau mai multe argumente și returnează o valoare pentru fiecare rând rezultat în urma interogării.

Există mai multe tipuri de funcții pe un singur rând: caracter, numerice, de date, de conversie etc. De exemplu, cele mai importante funcții caracter sunt:

```
UPPER(<expresie>)
```

UPPER convertește alfa caracterele din caractere mici în caractere mari.

```
LOWER (<expresie>)
```

LOWER convertește alfa caracterele din caractere mari în caractere mici.

```
SUBSTRING(<expresie>, m [, n])
```

SUBSTRING returnează un șir de n caractere începând cu caracterul aflat pe poziția m .

`LEN (<expresie>)`

LEN returnează numărul de caractere dintr-o expresie.

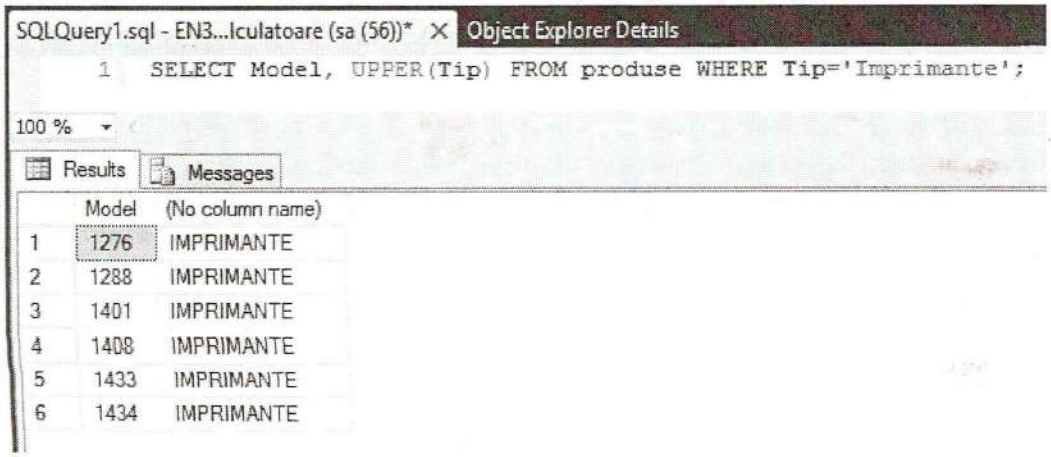
`REPLACE(<text>, <şir_căutare>, <şir_înlocuire>)`

REPLACE caută un anumit text într-un şir de caractere şi, dacă îl găseşte, îl înlocuieşte.

`CONCAT (<expresie1>, <expresie2>)`

CONCAT funcția este echivalentul operatorului de concatenare (+).

La început, în calitate de exemplu practic, este examinată o funcție de tipul ***Single row***. Fie este formulată o interogare, care solicită prezentarea cu majuscule a coloanei ***Tip*** din tabelul ***produse***. Pentru aceasta, este utilizată funcția *UPPER:*



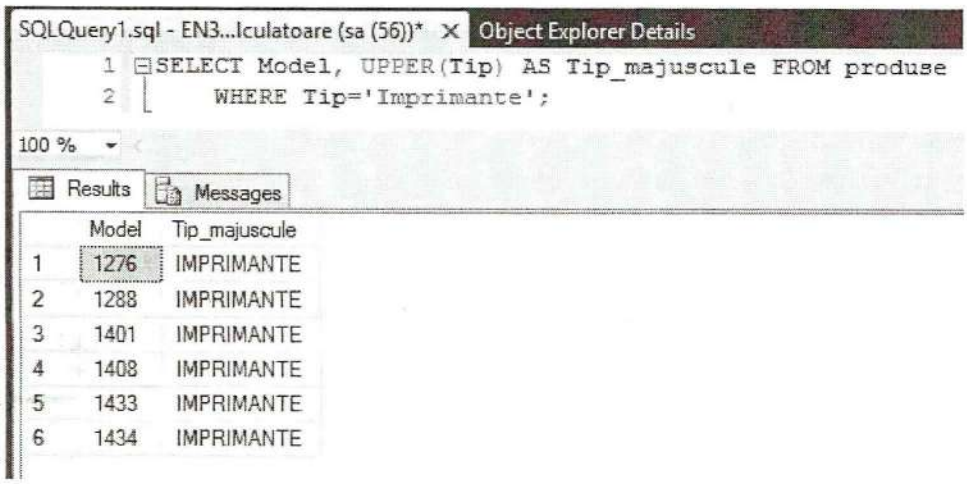
	Model	(No column name)
1	1276	IMPRIMANTE
2	1288	IMPRIMANTE
3	1401	IMPRIMANTE
4	1408	IMPRIMANTE
5	1433	IMPRIMANTE
6	1434	IMPRIMANTE

Să se observe că, în acest exemplu, este o singură noutate: înainte de câmpul ***Tip***, s-a scris funcția *UPPER*, care transformă caracterele minuscule în caractere majuscule. În afară de aceasta, în gridul interogării, câmpul care prezintă imprimantele nu are un nume.

În clauza *SELECT*, se pot redenumi câmpuri sau se pot specifica nume pentru expresii, folosind următoarea sintaxă:

`...<expresie> AS <nume expresie>...`

Se observă că noul nume atribuit unei coloane sau expresii urmează vechiului nume sau expresiei, precedat de cuvântul-cheie *AS*. Îndeosebi, acesta este un lucru obişnuit, fiindcă, după funcții, trebuie să fie indicat numele coloanei, folosind clauza *AS*, precum este prezentat în următorul exemplu:



	Model	Tip_majuscule
1	1276	IMPRIMANTE
2	1288	IMPRIMANTE
3	1401	IMPRIMANTE
4	1408	IMPRIMANTE
5	1433	IMPRIMANTE
6	1434	IMPRIMANTE

Crearea unui nou câmp calculat pe baza câmpurilor din tabelele implicate în interogare se realizează, în exemplul precedent, prin numele `Tip_majuscule`.

4.3.4. Funcții de grup

Funcțiile de grup (agregare) prelucrează mulțimi de înregistrări (linii) ale tabelului şi returnează un anumit rezultat. Majoritatea acestor funcții, de obicei, acționează asupra tuturor liniilor tabelului. Ele acceptă următorii parametri:

- *ALL* – face ca funcția să considere toate înregistrările, inclusiv cele duplicate. Această opțiune este implicită.
- *DISTINCT* – face ca funcția să considere numai înregistrările care nu posedă o valoare duplicată (în câmpurile specificate).

Din funcțiile de agregare, fac parte:

```
SUM ([DISTINCT | ALL] <expresie>)
```

Funcția *SUM* calculează suma unei mulțimi de valori numerice.

```
AVG ([DISTINCT | ALL] <expresie>)
```

Funcția *AVG* returnează media aritmetică a expresiei `<expresie>`.

```
{MAX | MIN} ([DISTINCT | ALL] <expresie>)
```

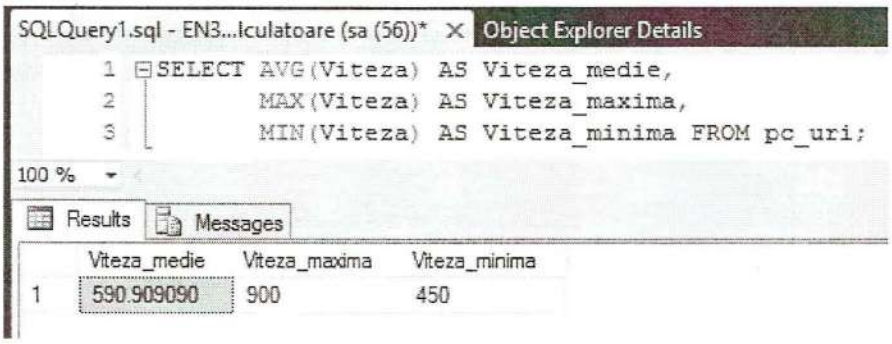
Funcția *MAX/MIN* returnează valoarea maximă sau minimă a expresiei.

```
COUNT (* | [DISTINCT | ALL] <expresie>)
```

Funcția *COUNT* returnează numărul de linii nenule returnate de interogare. Aici, `<expresie>` reprezintă un câmp sau expresie.

SQL permite sintetizarea datelor din baza de date prin utilizarea unor funcții de grup asupra coloanelor. O funcție de grup primeşte ca argument o întreagă coloană (sau coloane) şi furnizează ca rezultat o singură valoare, care sintetizează datele din coloana respectivă. Evident că instrucțiunile *SELECT* cu funcții bloc sunt mai complexe.

Se examinează un exemplu care utilizează 3 funcții: pentru a obține valoarea medie a unei coloane numerice, pentru a obține valoarea maximă dintr-o coloană şi pentru a obține valoarea minimă dintr-o coloană.



În acest exemplu, SGBD-ului i se solicită să calculeze viteza medie a microprocesoarelor, dar şi valoarea maximă înregistrată a vitezei, şi valoarea minimă înregistrată a vitezei microprocesoarelor, făcând uz de coloanele logice ***Viteza_medie***, ***Viteza_maxima***, ***Viteza_minima***, respectiv.

Funcția *COUNT* este uşor diferită de celelalte şi are 3 formate.

`COUNT(*)` întoarce numărul de rânduri dintr-un tabel care satisfac criteriul de selecție, incluzând rândurile duplicat şi rândurile ce conțin valori nule. În exemplul de mai jos, structura instrucțiunii este mult mai simplă: este solicitat, în calitate de rezultat, numărul de rânduri ale tabelului ***pc_uri***.

```
SELECT COUNT(*) FROM pc_uri
```

În contrast `COUNT(<expresie>)` întoarce numărul de valori nenule din coloana specificată de `<expresie>`. Interogarea din următorul exemplu solicită numărul total de linii în care valoarea câmpului ***Ram*** este diferită de valoarea *NULL*.

```
SELECT COUNT(Ram) FROM pc_uri
```

Această funcție, `COUNT(DISTINCT <expresie>)`, returnează numărul de valori distincte, nenule din coloana specificată de `<expresie>`. Calcularea numărului de linii, unde câmpul ***Ram*** are valori distincte, este prezentată de interogarea de mai jos:

```
SELECT COUNT(DISTINCT Ram) FROM pc_uri
```

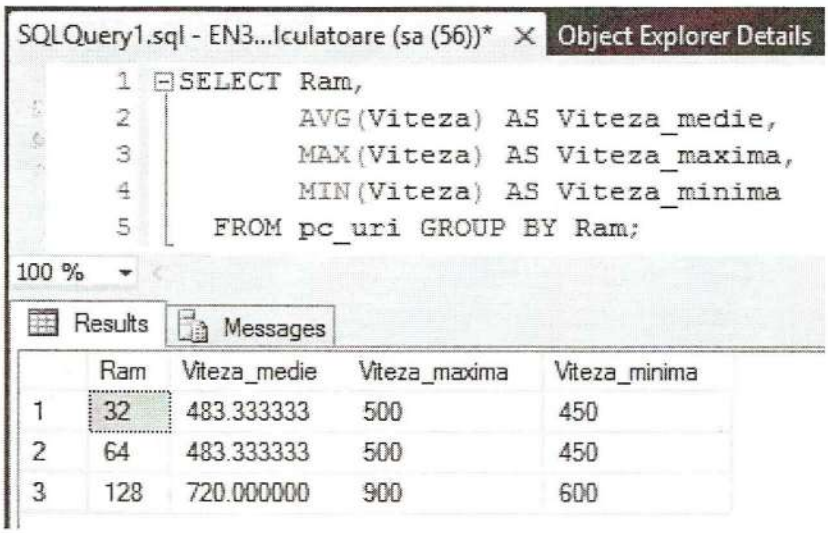
Până acum, toate funcțiile de grup au fost aplicate întregului tabel. Dar se poate împarți tabelul în grupuri mai mici. Folosirea acestuia returnează informații sumare despre fiecare grup. Sintaxa instruiților *SELECT* cu funcții de grup pot folosi clauzele *GROUP BY* sau *HAVING*:

```
SELECT [ALL|DISTINCT] <listă coloane>
       FROM <listă tabele>
       [WHERE <condiție căutare>]
       [GROUP BY <listă coloane>
       [HAVING <caracteristică_grup>]]
       [ORDER BY <listă coloane> [ASC | DESC] ]
```

Atunci când se apelează la funcții agregate în clauza *SELECT* se utilizează, de obicei, şi clauza *GROUP BY* pentru a separa informația în grupuri distincte.

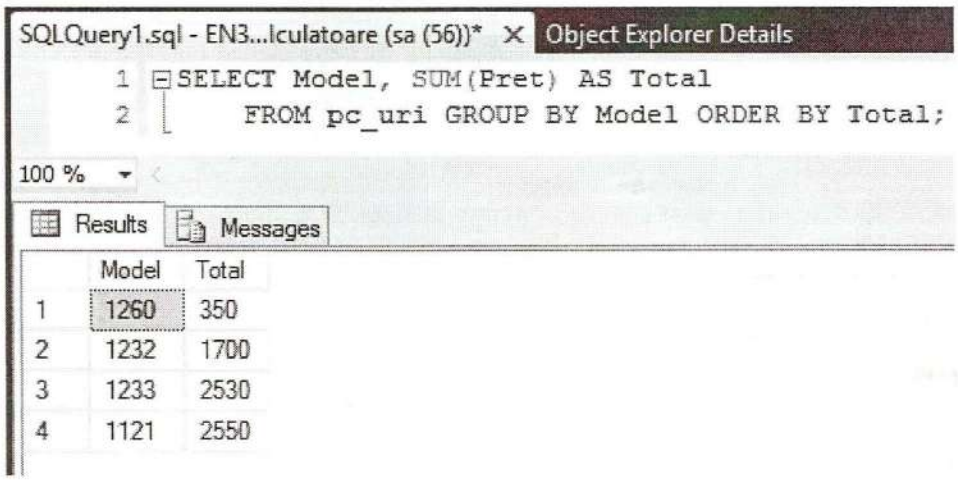
Clauza *GROUP BY* se foloseşte pentru a grupa rezultatele funcțiilor de grup după valoarea uneia sau mai multor coloane. Dacă se dorește calculul unei valori pe grupe de linii, atunci se introduce clauza *GROUP BY*, urmată de numele uneia sau mai multor coloane. În acest caz, funcția se aplică separat acelor linii care au aceeaşi valoare a coloanelor listate de clauza *GROUP BY*.

Astfel, în exemplul de mai jos, sunt prezentate valorile câmpului ***Ram*** (memorie operativă), în acelaşi timp, este calculată viteza medie, viteza minimă şi maximă, grupate după capacitatea memoriei, ***Ram***.



	Ram	Viteza_medie	Viteza_maxima	Viteza_minima
1	32	483.333333	500	450
2	64	483.333333	500	450
3	128	720.000000	900	600

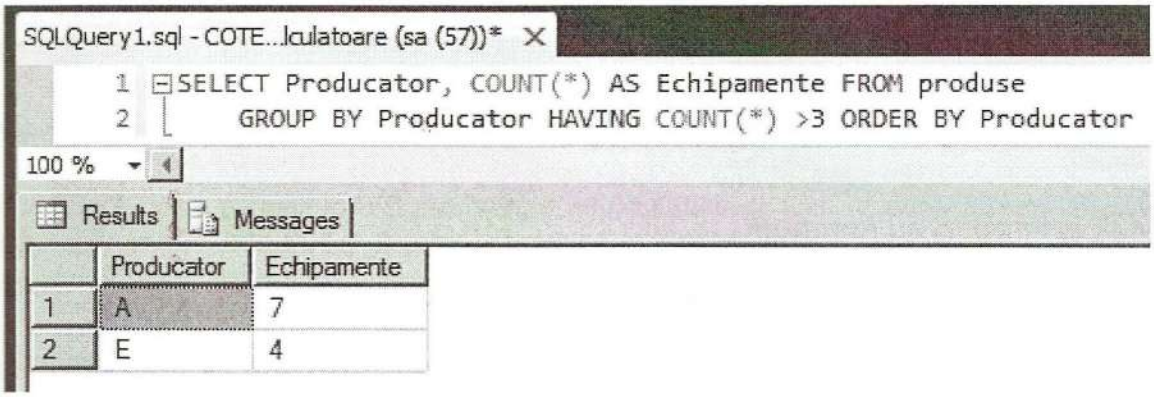
Se afişeze o interogare mai complexă. Interogarea calculează sumei totale a costului ***pc_urilor*** *(SUM(**Pret**))*, prezentând această sumă în câmpul logic numit ***Total***, grupată după modelul ***pc_urilor*** (***Model***) şi ordonate pe câmpul ***Total***. Să se observe rezultatul acestei interogări:



Clauza *HAVING* operează la nivel de grupuri ce au fost create, în prealabil, cu clauza *GROUP BY*. Clauza *HAVING* este asemănătoare clauzei *WHERE*. *WHERE* acționează asupra câmpurilor ce îndeplinesc condiția specificată, în timp ce *HAVING* acționează asupra grupurilor.

Cu ajutorul clauzei *HAVING*, se pot pune condiții pe grupurile de linii asociate cu fiecare valoare a coloanelor de grupare. Doar grupurile care satisfac condiția sunt afișate în rezultatul interogării.

De exemplu, pentru fiecare producător, se afișează numărul de echipamente și numărul de echipamente realizate de fiecare producător e mai mare decât 3. Doar grupurile care satisfac condiția (> 3) sunt afișate în rezultatul interogării. Ordonarea se face în mod crescător.



Trebuie menționat că clauza *HAVING* trebuie să preceadă întotdeauna clauza *ORDER BY*.

4.3.5. Limitarea numărului de linii returnate

În SQL Server 2014, clauza *TOP* permite extragerea unui număr sau un procent de linii returnate de o interogare. Atunci când *TOP* se utilizează împreună cu clauza *ORDER BY*, mulțimea întoarsă este mărginită la primele N linii ordonate; în caz contrar, sunt returnate primele N rânduri în ordine nedefinit. Clauza *TOP* este disponibilă pentru instrucțiunile *SELECT*, *INSERT*, *UPDATE*, *MERGE* sau *DELETE*.

Acestea folosesc următoarea sintaxă:

```
[ TOP <expresie> [PERCENT] [WITH TIES] ]...
```

`<expresie>`

Reprezintă numărul de linii returnate. Este reprezentată sub formă de întreg (*BIGINT*) sau în procentaj de linii întoarse. Procentajul poate fi exprimat sub formă *FLOAT*.

`[PERCENT]`

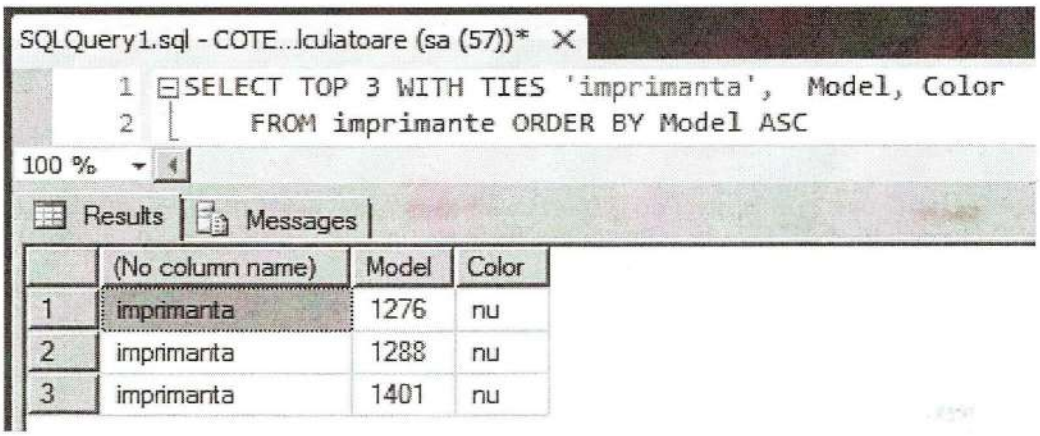
Specifică faptul că numărul este reprezentat în procente.

```
[WITH TIES]
```

Selectarea liniilor se efectuează după trierea datelor de către clauza *ORDER BY*.

TOP...WITH TIES poate fi definită, în mod univoc, numai în instrucțiunea *SELECT*, şi numai dacă este menționată clauza *ORDER BY*. Ordinea returnării înregistrărilor legate sunt arbitrare (dacă câmpurile sortate au duplicate). *ORDER BY* nu afectează această regulă.

Urmează un exemplu, care prezintă modul în care SQL Server limitează numărul de tupluri returnate. Presupunem că dorim să extragem primele 3 înregistrări din tabelul ***imprimante,*** sortate ascendent conform interogării:.



Deci, sortarea este executată, în mod crescător, după câmpul ***Model***. În setul de rezultate, vor fi 3 (*TOP* 3) linii. Cu toate acestea, există o problemă în cazul în care mai multe imprimantă din tabel vor avea aceleaşi model. Problema este rezolvată cu ajutorul *WITH TIES*, care include în setul de rezultate şi restul care au rămas cu valori ce coincid cu cele 3 imprimante inițiale.

În cazul nostru rezultatul final va rămâne 3, deoarece nu mai sunt în restul modelelor identice cu cele 3 inițiale.

4.4. Interogări cu joncțiuni

Cea mai folosită instrucțiune SQL pentru obținerea informațiilor din mai multe tabele relaționate este *JOIN*, sau joncțiunea. Instrucțiunile *JOIN* pot fi interne sau externe.

Deseori, pentru a obține un rezultat complet, este necesară selectarea datelor din două sau mai multe tabele. De obicei, tabelele din baza de date sunt legate între ele prin chei. Este important, îndeosebi, rolul cheii primare care face ca coloanele ce o constituie să posede o valoare unică pentru fiecare rând.

De exemplu, este necesar să se afişeze modelele, producătorul şi viteza laptopurilor. Câmpul ***Viteza*** se află în tabelul ***laptop_uri***. Informația referitoare la producători se află în tabelul ***produse***, rezultatul este ordonat după câmpul ***Producator***. Nu este afişat doar producătorul `'A'`.

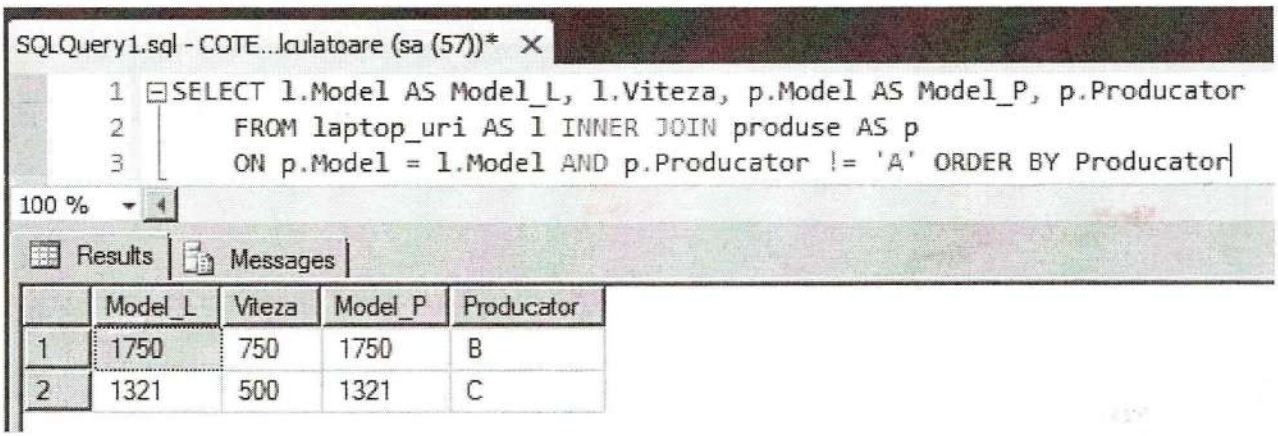
```
SELECT l.Model AS Model_L, l.Viteza,
       p.Model AS Model_P, p.Producator
    FROM laptop_uri AS, l produse AS p
    WHERE p.Model=l.Model AND p.Producator != 'A'
    ORDER BY Producator
```

Pentru *JOIN*, sistemul SQL Server oferă şi o sintaxă specifică, introdusă de către standardul SQL3 (SQL:1999). Această sintaxă nu aduce beneficii în privința performanței față de joincțiunile care se aplică în clauza *WHERE*. Tipurile de joincțiuni conforme cu SQL3 sunt definite prin cuvintele cheie *INNER JOIN, LEFT* şi *RIGHT [OUTER] JOIN, CROSS JOIN* (pentru produs cartezian), şi clauza *ON*.

4.4.1. Interogări cu joncțiuni interne

Joncțiunile interne, în mare măsură, corespund theta-joncțiunilor din algebra relațională. Joncțiunile interne returnează liniile din tabelele asociate, care satisfac criteriile de asociere prin compararea valorilor coloanelor respective din cele două tabele. Liniile care nu satisfac criteriile sunt eliminate.

Același exercițiu poate fi realizat, utilizând joncțiunea internă: *INNER JOIN* returnează toate rândurile din ambele tabele, care coincid pe câmpurile de joncțiune, cu excepția producătorului `'A'`. Dacă sunt rânduri în ***laptop_uri***, care nu se potrivesc cu rânduri din ***produse***, aceste rânduri nu vor figura în rezultat.



Instrucțiunea *SELECT* cu *INNER JOIN* are următoarea formă generală:

```
SELECT expresie1 [, expresie2 ...]
    FROM tabel_1 INNER JOIN tabel_2
    [ON <condiții joncțiune>]
```

Condiția `<condiții joncțiune>` efectuează operații pe baza condiției exprimate în clauza *ON*. Această clauză permite specificarea separată a condițiilor de join, respectiv a celor de căutare sau filtrare (din clauza *WHERE*).

4.4.2. Interogări cu joncțiuni externe

Sintaxa corespunzătoare prezenta forma generală a joncțiunii externe:

```
SELECT expresie1 [, expresie2 ...]
    FROM tabel_1 {{LEFT|RIGHT|FULL} [OUTER]} JOIN tabel_2
    [ON <condiții joncțiune>]
```

Spre deosebire de joncțiunile interne, unde opțional se folosea cuvântul-cheie *INNER*, joncțiunea externă utilizează opțional cuvântul-cheie *OUTER*. Cuvântul *OUTER* este unul opțional și nu adaugă nici o funcție.

Pentru realizarea unei joncțiuni externe, în clauza *FROM*, se indică numele tabelelor separate de operatorul *OUTER JOIN*, cu folosirea uneia din variantele *LEFT*, *RIGHT* sau *FULL*. Utilizarea clauzei *ON* permite specificarea condițiilor de joncțiune separat de condițiile de căutare sau filtrare din clauza *WHERE*.

Pentru construirea joncțiunii externă la stânga (*LEFT OUTER JOIN*), la început se realizează o joncțiune internă. Apoi, fiecare rând din tabelul *T1* (*T1* stânga, *T2* dreapta) care nu satisface joncțiunea cu *T2* este adăugat având valori *NULL* pentru câmpurile corespondente din *T2*. Tabelul rezultat va avea, cel puțin rândurile din tabelul *T1*.

LEFT [OUTER] JOIN returnează toate rândurile din primul tabel (***laptop_uri***), chiar dacă nu coincid pe câmpurile de joncțiune cu vreun rând din al doilea tabel (***produse***). Adică, dacă există așa tupluri în tabelul ***laptop_uri***, care nu coincid pe câmpurile de joncțiune din tabelul ***produse***, acestea sunt, de asemenea, afișate. Coloanele tabelului din dreapta se completează cu valori nule.

SQLQuery1.sql - COTE...lculatoare (sa (57))* ×

```
1  SELECT l.Model AS Model_L, l.Viteza, p.Model AS Model_P, p.Producator
2      FROM laptop_uri AS l  LEFT JOIN  produse AS p
3      ON p.Model = l.Model AND p.Producator != 'A' ORDER BY Producator
```

100 %

Results | Messages

	Model_L	Viteza	Model_P	Producator
1	1298	350	NULL	NULL
2	1298	600	NULL	NULL
3	1752	750	NULL	NULL
4	1298	450	NULL	NULL
5	1750	750	1750	B
6	1321	500	1321	C

Pentru construirea joncțiunii externă la dreapta (*RIGHT OUTER JOIN*). la început se realizează o joncțiune internă. Apoi, fiecare rând din tabelul *T2* (*T1* stânga, *T2* dreapta) care nu satisface joncțiunea cu *T1* este adăugat având valori *NULL* pentru câmpurile corespondente din *T1*. Tabelul rezultat va avea, cel puțin rândurile din tabelul *T2*.

RIGHT [OUTER] JOIN returnează toate rândurile din al doilea tabel (***produse***), chiar dacă nu coincid pe câmpurile de joncțiune cu rânduri din primul tabel (***laptop_uri***). Adică, dacă există asemenea tupluri în tabelul ***produse***, care nu coincid pe câmpurile de joncțiune din tabelul ***laptop_uri***, acestea sunt, de asemenea, afișate. Coloanele tabelului din stânga se completează cu valori nule.

SQLQuery1.sql - COTE...lculatoare (sa (57))* ×

```
1  SELECT l.Model AS Model_L,
2         l.Viteza,
3         p.Model AS Model_P,
4         p.Producator
5      FROM laptop_uri AS l  RIGHT JOIN  produse AS p
6      ON p.Model = l.Model AND p.Producator != 'A'
7      ORDER BY Producator
```

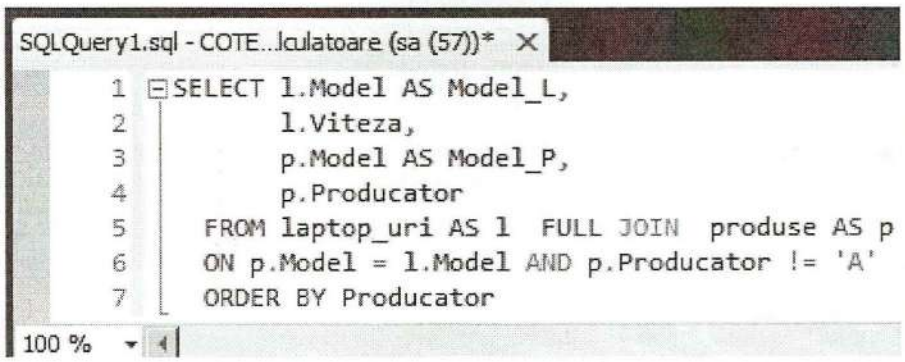
100 %

Results | Messages

	Model_L	Viteza	Model_P	Producator
1	NULL	NULL	1232	A
2	NULL	NULL	1233	A
3	NULL	NULL	1276	A
4	NULL	NULL	1401	A
5	NULL	NULL	1408	A
6	NULL	NULL	1298	A
7	NULL	NULL	1752	A
8	NULL	NULL	1121	B
9	1750	750	1750	B
10	1321	500	1321	C
11	NULL	NULL	1433	D
12	NULL	NULL	1288	D
13	NULL	NULL	1260	E
14	NULL	NULL	1434	E
15	NULL	NULL	2111	E
16	NULL	NULL	2112	E

Pentru construirea joncțiunii externă totale (*FULL OUTER JOIN*). la început se realizează o joncțiune internă. Apoi, fiecare rând din tabelul *T2* (*T1* stânga, *T2* dreapta) care nu satisface joncțiunea cu *T1* este adăugat având valori *NULL* pentru câmpurile corespondente din *T1*. Apoi, fiecare rând din *T1* care nu satisface joncțiunea cu *T2* este adăugat având valori NULL pentru câmpurile corespondente din *T2*.

FULL [OUTER] JOIN returnează toate rândurile din al doilea tabel (***produse***), chiar dacă nu coincid pe câmpurile de joncțiune cu rânduri din primul tabel (***laptop_uri***). Adică, dacă există asemenea tupluri în tabelul ***produse***, care nu coincid pe câmpurile de joncțiune din tabelul ***laptop_uri***, acestea sunt, de asemenea, afișate. Apoi returnează toate rândurile din primul tabel (***laptop_uri***), chiar dacă nu coincid pe câmpurile de joncțiune cu vreun rând din al doilea tabel (***produse***). Adică, dacă există așa tupluri în tabelul ***laptop_uri***, care nu coincid pe câmpurile de joncțiune din tabelul ***produse***, acestea sunt, de asemenea, afișate. Coloanele tabelelor din stânga li din dreapta se completează cu valori nule.



```
SQLQuery1.sql - COTE...lculatoare (sa (57))* X
1  SELECT l.Model AS Model_L,
2         l.Viteza,
3         p.Model AS Model_P,
4         p.Producator
5    FROM laptop_uri AS l  FULL JOIN  produse AS p
6    ON p.Model = l.Model AND p.Producator != 'A'
7    ORDER BY Producator
100 %
```

Results | Messages

	Model_L	Viteza	Model_P	Producator
1	1298	600	NULL	NULL
2	1752	750	NULL	NULL
3	1298	450	NULL	NULL
4	1298	350	NULL	NULL
5	NULL	NULL	1232	A
6	NULL	NULL	1233	A
7	NULL	NULL	1298	A
8	NULL	NULL	1401	A
9	NULL	NULL	1408	A
10	NULL	NULL	1276	A
11	NULL	NULL	1752	A
12	NULL	NULL	1121	B
13	1750	750	1750	B
14	1321	500	1321	C
15	NULL	NULL	1288	D
16	NULL	NULL	1433	D
17	NULL	NULL	1434	E
18	NULL	NULL	2111	E
19	NULL	NULL	2112	E
20	NULL	NULL	1260	E

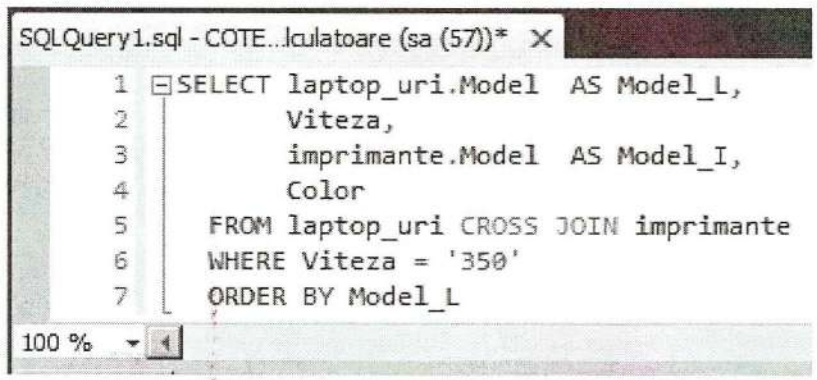
4.4.3. Produsul Cartezian

Cross join (sau joncțiunea încrucișată) este produsul Cartezian al tabelelor implicate. În acest fel, pentru fiecare rând din primul tabel se adaugă fiecare rând din al doilea tabel. Rezultatul are lungimea produsului numărului de linii din cele două tabele.

Sintaxa corespunzătoare prezenta forma generală a produsului Cartezian:

```
SELECT expresie1 [, expresie2 ...]
    FROM tabel_1 CROSS JOIN tabel_2
```

De exemplu, se returnează un tabel ce conține produsul Cartezian cu toate rândurile din tabelele ***laptop_uri*** și ***imprimante***. Coloanele din noul tabel sunt coloanele ***laptop_uri.Model***, ***Viteză***, ***imprimante.Model*** și ***Color*** din toate cele două tabelele. Este acceptată doar ***Viteza*** = `'350'`.



```
SQLQuery1.sql - COTE...lculatoare (sa (57))* X
1  SELECT laptop_uri.Model  AS Model_L,
2         Viteza,
3         imprimante.Model  AS Model_I,
4         Color
5    FROM laptop_uri CROSS JOIN imprimante
6    WHERE Viteza = '350'
7    ORDER BY Model_L
100 %
```

Results | Messages

	Model_L	Viteza	Model_I	Color
1	1298	350	1276	nu
2	1298	350	1433	da
3	1298	350	1434	da
4	1298	350	1401	nu
5	1298	350	1408	nu
6	1298	350	1288	nu

Instrucțiunea de mai sus este echivalentă cu instrucțiunea:

```
SELECT laptop_uri.Model AS Model_L, Viteza,
       imprimante.Model AS Model_I, Color
    FROM laptop_uri, imprimante
    WHERE Viteza = '350' ORDER BY Model_L
```

Produsul Cartezian a două tabele asociază fiecare rând din primul tabel cu toate rândurile din tabelul al doilea. El conservă toate rândurile tuturor tabelelor joncționate, obținând toate combinațiile posibile ale acestora. Desigur, dacă nu există duplicate în tabelele originale, toate rândurile din rezultat vor fi, de asemenea, unice.

Un produs Cartezian are tendința de a genera un număr mare de rânduri, iar rezultatul, de regulă, rar este util. Acesta poate parveni accidental în cazul nespecificării criteriilor de joncționare, în particular, atunci când se utilizează sintaxa SQL89. Prin urmare, deseori, trebuie să fie prezentă o condiție de valabilitate a joncțiunii, fapt ce transformă produsul Cartezian într-un alt tip de joncțiune.

Excepție fac doar cazurile în care este nevoie de combinarea tuturor rândurilor din toate tabelele. Produsul Cartezian poate fi folosit, de asemenea, pentru executarea unor teste în cazul în care este necesară generarea unui număr mare de rânduri pentru simularea unui număr rezonabil de date.

4.5. Subinterogări şi operatori pe mulțimi

O subinterogare este o interogare inclusă într-o altă interogare. Rezultatul subinterogării este utilizat de SGBD pentru a determina rezultatele interogării de nivel mai înalt care conține subinterogarea. În forma sa cea mai simplă subinterogarea apare în clauza *WHERE*. Dar pot exista şi subinterogări în clauzele *FROM*, *HAVING* şi în alte clauze.

În afară de acesta, operatorii definiți pe mulțimi permit obținerea unei interogări a mai multor tabele combinând rezultatele a două sau mai multe interogări independente una de cealaltă.

4.5.1. Subinterogări corelate şi necorelate

Subinterogările mai sunt numite instrucțiuni *SELECT* imbricate. Utilizarea blocurilor imbricate permite exprimarea simplă şi într-un mod foarte clar a expresiilor cuantificate. Un bloc intern, de obicei, se numeşte subinterogare. O subinterogare este o cerere care, de regulă, restituie un singur câmp. Prin intermediul subinterogărilor se pot construi interogări complexe pe baza unor instrucțiuni simple.

O subinterogare (subcerere) este o instrucțiune *SELECT* integrată într-o clauză a altei instrucțiuni SQL, numită instrucțiune exterioară sau instrucțiune „părinte". Prin urmare, subinterogările sunt instrucțiuni interioare.

Rezultatele subinterogării sunt utilizate în cadrul cererii exterioare, pentru a determina rezultatul final. În funcție de modul de evaluare a subinterogării în raport cu cererea exterioară, subinterogările pot fi: subinterogări necorelate (nesincronizate) sau corelate (sincronizate).

Prima clasă de subinterogări este evaluată dinspre interior către exterior, adică interogarea externă acționează pe baza rezultatului cererii interne. Subinterogările necorelate care apar în clauza *WHERE* a unei interogări sunt de forma următoare:

```
SELECT expresie1 [, expresie2 ...] FROM tabel_1
    WHERE <expresie_condiție> <operator>
    (SELECT expresie FROM tabel_2);
```

Paşii de execuție ai unei subinterogări corelate:

- cererea internă este executată prima şi determină o valoare (sau o mulțime de valori);
- cererea externă se execută o singură dată, utilizând valorile returnate de cererea internă.

Al doilea tip de subinterogare este evaluat invers, adică interogarea externă furnizează valori cererii interne, iar rezultatele subinterogării sunt transferate cererii externe. Subinterogările corelate care apar în clauza *WHERE* a unei interogări au următoarea formă generală:

```
SELECT expresie_ext_1 [,expresie_ext_2 ...] FROM tabel_1_ext
    WHERE <expresie_condiție> <operator>
    (SELECT expresie FROM tabel_2
        WHERE expresie = tabel_1_ext.expresie_ext);
```

Paşii de execuție ai unei subinterogări corelate:

- cererea externă determină un rând candidat;
- cererea internă este executată utilizând valoarea rândului candidat;

- valorile rezultate din cererea internă sunt utilizate pentru calificarea sau descalificarea rândului candidat;
- paşii precedenți se repetă până când nu mai există rânduri candidat.

Deşi subinterogarea corelata se execută repetat, o data pentru fiecare linie în cererea principală, aceasta nu înseamnă că subinterogările corelate sunt mai puțin eficiente ca subcererile necorelate obişnuite.

Din punct de vedere al numărului de linii şi coloane returnate, există trei tipuri de subinterogări:

- scalare, care returnează o singură coloană şi o singură linie şi pot fi folosite atunci când este necesară o singură valoare;
- pe o linie, care returnează coloane multiple, dar o singură linie, şi pot fi utile atunci când este necesar un constructor de valori ale liniei (de regulă, în predicate);
- pe un tabel, care returnează una sau mai multe coloane şi linii multiple

Subinterogarea se leagă cu interogarea principală folosind operatorii. Operatorul ce intervine în specificarea condiției poate fi de tip:

- operator *single row* (>, =, >=, <, <>, <=), care poate fi utilizat dacă subinterogarea returnează o singură linie;
- operator *multiple row* (*IN, ANY, ALL*), care poate fi folosit dacă subinterogarea returnează mai mult de o linie.
- operator *EXISTS,* care poate fi folosit dacă subinterogarea returnează un boolean

Operatorul *NOT* poate fi utilizat în combinație cu *IN, ANY, ALL* şi *EXISTS.*

Cuvintele-cheie *ANY* şi *ALL* pot fi utilizate cu subinterogările care produc o singură coloană de valori. Dacă subinterogarea este precedată de către cuvântul-cheie *ALL*, atunci condiția va fi adevărată numai dacă este satisfăcută de către toate valorile produse de subinterogare. Astfel, *<ALL* are semnificația „mai mic decât minimul", iar *>ALL* este echivalent cu „mai mare decât maximul". Dacă subinterogarea este precedată de către cuvântul-cheie *ANY*, condiția va fi adevărată dacă este satisfăcută de către oricare (una sau mai multe) dintre valorile produse de subinterogare. În comparații, *<ANY* are semnificația „mai mic decât maximul"; *>ANY* înseamnă „mai mare decât minimul"; *=ANY* este echivalent cu operatorul *IN.*

Dacă subinterogarea returnează mulțimea vidă, atunci condiția *ALL* va returna valoarea *TRUE,* iar condiția *ANY* va returna valoarea *FALSE.* Cuvântul-cheie *EXISTS* este urmată de o subinterogare între paranteze şi ia valoarea *TRUE,* dacă există cel puțin un tuplu care satisface condițiile subinterogării. Iar în cazul când subinterogare nu întoarce nicio linie, rezultatul predicatului *EXISTS* este *FALSE.* Standardul ISO permite utilizarea cuvântului-cheie *SOME,* în locul lui *ANY.*

4.5.2. Subinterogări ce întorc un scalar sau o linie

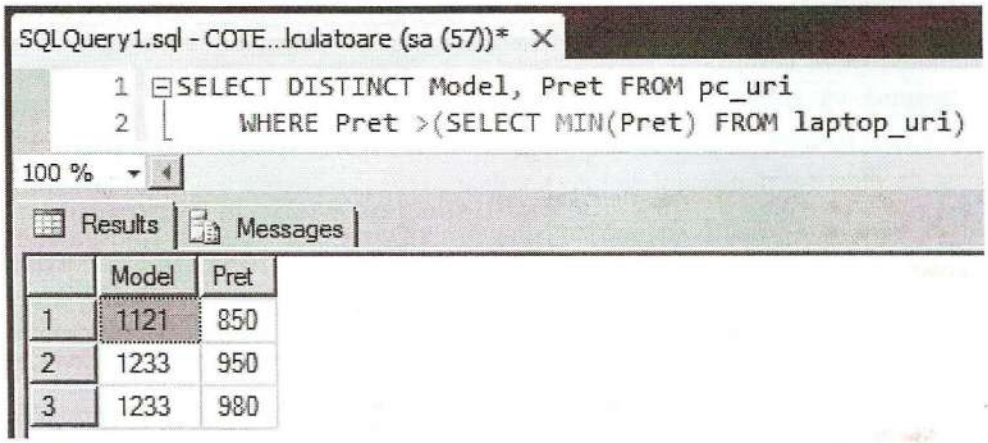
Există subinterogări care, în mod obligatoriu, trebuie să întoarcă un scalar sau o singură linie. Acestea folosesc următoarea sintaxă:

```
...{expresie|(expresie1 [, expresie2 ...])}
      {>|<|=|>=|<=|<>} (<subinterogare>)
```

Să se observe că interogarea conține, în afară de cei doi operanzi, un operator. Operatorul ce intervine în specificarea condiției reprezintă un operator de comparație (=, >, >=, <, <=, <>) care se aplică numai în cazul în care subinterogarea returnează o singură valoare sau o singură linie de valori (definită pe una sau mai multe coloane).

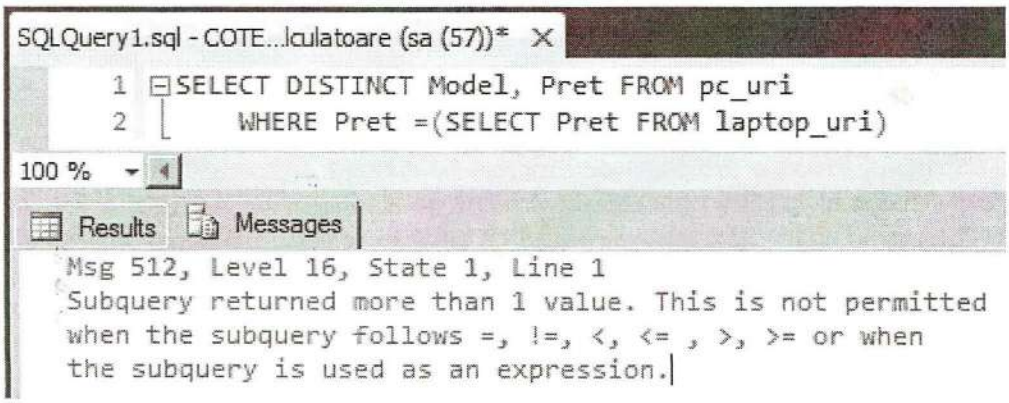
Predicatul este evaluat cu valoarea *TRUE,* în cazul în care comparația (indicată de unul din operatorii de comparație) rezultatului expresiei (expresiilor) cu cel returnat de subinterogare, este *TRUE.* În caz contrar, se evaluează la *FALSE.* Dacă interogarea subordonată nu produce nicio valoare, predicatul se evaluează la valoarea *NULL.* Dacă subinterogarea returnează mai mult de o valoare (mai multe linii definite pe o coloană, sau cel puțin o linie definită pe mai multe coloane) se produce o eroare de executare.

Următorul exemplu de cod reprezintă o subinterogare scalară, care determină modelele şi prețurile pc-urilor procurate la prețuri mai mari decât prețul minimal al laptopurilor:



Această interogare este destul de corectă, deoarece valoarea scalară a prețului se compară cu subinterogare care returnează o singură valoare. Ca rezultat, sunt obținute trei modele de pc-uri.

Cu toate acestea, în cazul în care se scrie următoarea interogare referitoare la întrebarea cu privire la modelele şi prețurile pc-urilor şi prețurile pc-urilor sunt aceleaşi cu cele ale laptopurilor:



Sistemul a afişat un mesaj de eroare, indicând faptul că pot fi obținute la subinterogare mai multe valori.

4.5.3. Subinterogări cu operatorul IN

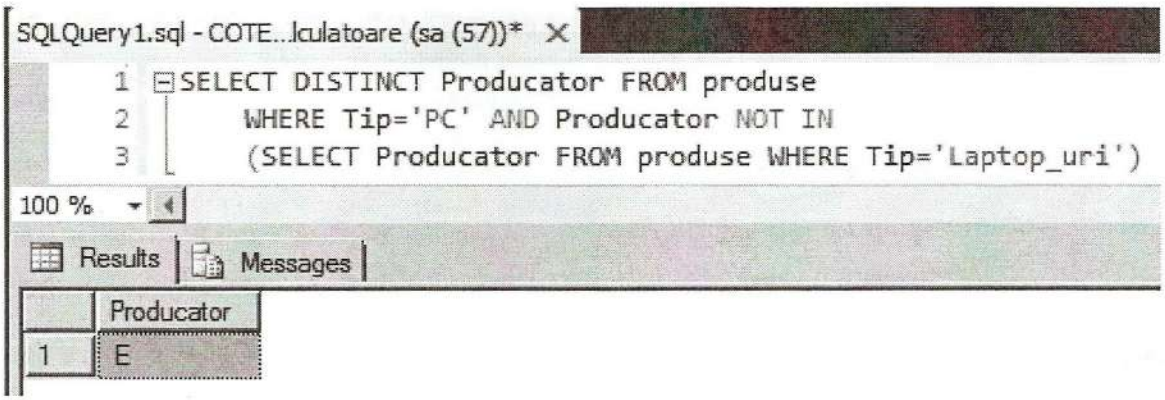
Operatorul *IN* verifică apartenența unei expresii (sau unei secvențe de expresii) la o mulțime de valori rezultată dintr-o subinterogare. Predicatul care include operatorul *IN* utilizat cu o interogare subordonată are următoarea formă generală:

```
...{expresie|(expresie1 [, expresie2 ...])}
                [NOT] IN (<subinterogare>)...
```

Din sintaxa generală, se observă că există două forme ale predicatului cu includerea operatorului *IN*. Predicatul, în forma sa mai simplă, presupune că, în partea stângă a operatorului *IN*, se poate găsi o expresie, iar a două variantă presupune că partea stângă este constituită dintr-o linie de expresii. Operatorul *NOT IN* verifică neapartenența la o mulțime.

Predicatul este evaluat la *TRUE*, dacă, în subinterogare, se găseşte vreo linie egală cu cel din stânga operatorului *IN*. În caz contrar, este evaluat la *FALSE* (chiar dacă subinterogarea nu întoarce nicio linie). Dacă subinterogarea întoarce unele linii nule, iar restul liniilor sunt distincte de linia din stânga operatorului *IN*, predicatul este evaluat la *NULL*.

O interogare imbricată poate fi o alternativă sintactică a unei expresii cu joncțiuni. În exemplul de mai jos, se cere să se găsească producătorii care vând ***pc-uri***, dar nu vând ***laptop-uri***.



În interogarea dată, în calitate de filtru, se utilizează negația tuturor elementelor conținute în tabelul rezultant al subinterogării.

4.5.4. Subinterogări cu operatorul ANY

Operatorul *ANY* poate modifica acțiunile operatorului de comparație pentru a permite interogării externe să accepte mai multe linii returnate de interogarea subordonată. Sintaxa generală a predicatului, care conține operatorul *ANY,* este următoarea:

```
...{expresie|(expresie1 [, expresie2 ...])}
    {>|<|=|>=|<=|<>} ANY (<subinterogare>)...
```

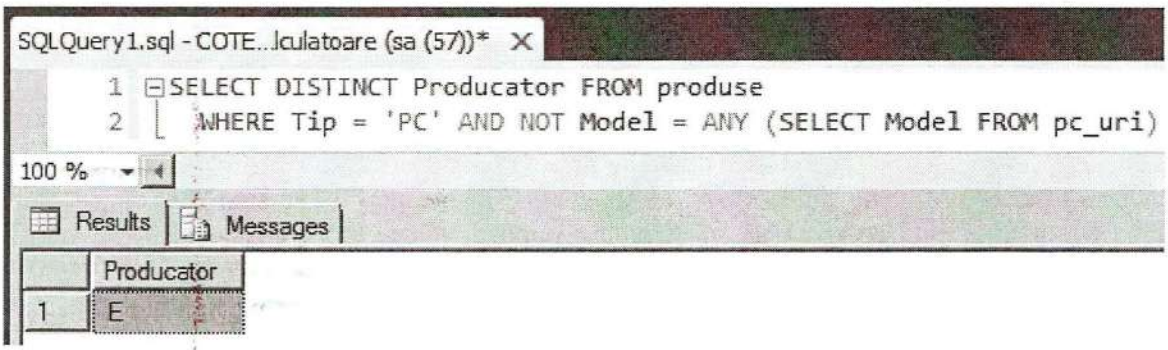
Cuvântul-cheie *ANY,* care trebuie să urmeze după un operator de comparație întoarce valoarea adevărat dacă este adevărată comparația pentru cel puțin unul dintre valorile pe care le întoarce subinterogarea.

Din cele ce urmează, reiese că subinterogarea nu întoarce numai o valoare de tip scalar sau de tip coloană, ci și de tip tabel. Subinterogările tip tabel sunt acelea care întorc mai multe linii și mai multe coloane.

Predicatul ia valoarea *TRUE,* dacă rezultatul comparației cu cel puțin o valoare construită de subcerere este *TRUE* și întoarce valoarea *FALSE,* în caz contrar (se include și cazul special al interogării, care nu returnează nicio linie). Dacă subinterogarea returnează o linie cu componente nule, predicatul nu poate fi *FALSE* (sau ia valoarea *TRUE,* sau ia *NULL*).

Operatorul *IN* este echivalent cu = *ANY.* Ca și în cazul lui *IN,* nu se poate presupune că subinterogarea este complet evaluată.

Exemplul următor folosește o subinterogare care determină toți producătorii de pc-uri ale căror modele nu sunt comercializate în prezent (adică, nu sunt disponibile în tabelul ***pc_uri***):



Întrucât este un singur rezultat, producătorul E nu a furnizat modele pentru vânzare.

Se consideră exemplul în detalii. Predicatul

```
Model = ANY (SELECT Model FROM pc_uri)
```

întoarce valoarea *TRUE,* dacă modelul specificat de coloana ***Model*** din interogarea principală va fi în lista de modele a tabelului pc_uri (returnată de subinterogare). Din cauza utilizării predicatului *NOT*, valoarea

TRUE va fi obținută cu excepția cazului dacă modelul este în listă. Acest predicat este verificat pentru fiecare linie din interogarea principală care returnează toate modelele de pc-uri (predicatul ***Tip*** = 'PC') în tabelul ***produse***. Mulțimea rezultat este formată dintr-o singură coloană - ***Producător***. Cuvântul-cheie *DISTINCT* este folosit pentru a elimina orice duplicate ale producătorului ce pot apărea când un producător poate produce mai mult de un model absent în tabelul ***pc_uri***.

4.5.5. Subinterogări cu operatorul ALL

Operatorul *ALL* compară o valoare cu toate valorile returnate de o subinterogare. Sintaxa generală de utilizare a acestui operator în predicatele de filtrare are forma:

```
...{expresie|(expresie1 [, expresie2 ...])}
   {>|<|=|>=|<=|<>} ALL (<subinterogare>)...
```

Comparația este adevărată daca ea este adevărată pentru orice valoare returnata de subinterogare.

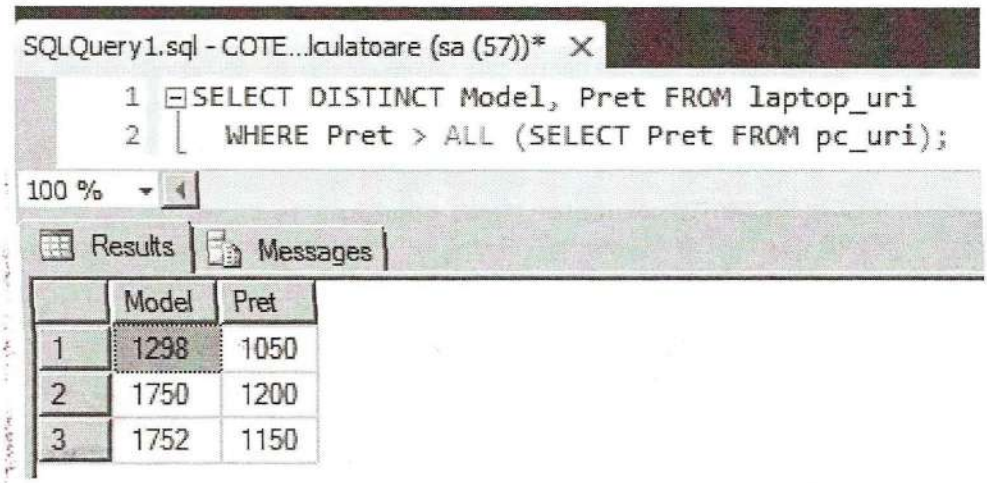
Partea stângă reprezintă o expresie sau o linie de expresii, iar partea dreaptă este o subinterogare în paranteze care returnează exact același număr și tip de coloane ca al expresiei sau ca al expresiilor ce constituie linia.

Mai întâi, va fi examinată situația când, în partea stângă, este un scalar, sau o expresie care este evaluată la un scalar:

În această utilizare a operatorului *ALL*, subinterogarea trebuie să returneze o singură coloană. Predicatul se evaluează la *TRUE*, dacă comparația stabilită de către operatorul de comparație este valabilă pentru toate valorile coloanei revenite de la interogarea subordonată. Predicatul se evaluează, de asemenea, la *TRUE* atunci când subinterogarea nu returnează nicio linie. În caz contrar, se evaluează la *FALSE*. În cazul în care subinterogarea returnează vreun *NULL*, predicatul se evaluează la *NULL*.

În utilizarea care urmează a operatorului *ALL*, în partea stângă este o linie. Expresiile din partea stângă sunt evaluate și comparate cu fiecare din liniile rezultatului, folosind operatorul de comparație dat. Rezultatul predicatului cu *ALL* este *TRUE*, dacă comparația returnează *TRUE* pentru toate liniile subinterogării (se include și cazul special al interogării care nu returnează nicio linie). Rezultatul este *FALSE*, dacă comparația are valoarea *FALSE* cel puțin pentru o linie returnată de interogarea subordonată. Rezultatul este *NULL*, dacă comparația nu are valoarea *FALSE* pentru vreo linie și returnează *NULL* pentru cel puțin o linie.

Fie se scrie următoarea interogare, dacă se dorește să se afle modelele și prețurile laptopurilor cu prețurile mai mari decât orice *pc*.



4.5.6. Subinterogări cu operatorul EXISTS

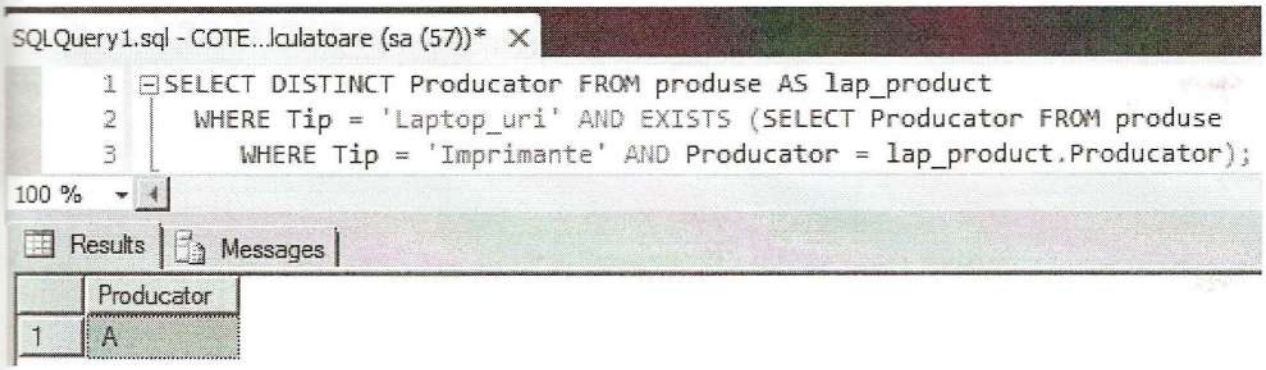
Majoritatea subinterogărilor prezentate mai sus au fost independente de interogarea exterioara (cea care le folosea). Put fi create însă și subinterogări corelate cu interogarea exterioară. Operatorul *EXISTS* este frecvent proiectat pentru a fi utilizat numai în contextul subinterogărilor. El este folosit cu subinterogările corelate și testează dacă ceva există sau nu există:

```
...[NOT] EXISTS (<subinterogare>) ...
```

Astfel, operatorul produce un rezultat simplu, de tip boolean. Condiția obținută cu ajutorul cuvântului-cheie *EXISTS* este *TRUE*, dacă există cel puțin o linie returnată de către subcerere, și este FALSE, dacă subinterogarea returnează o mulțime rezultat vidă. Acest operator este util pentru a testa dacă valoarea recuperată de cererea externă există în mulțimea valorilor regăsite de o cerere internă corelată a sa.

Deoarece operatorul *EXISTS* nu face decât să verifice existența sau inexistența liniilor în rezultatul subinterogării, aceasta poate conține orice număr de coloane. Nefiind necesar ca subinterogarea să returneze o anumită valoare, se poate selecta o constantă. De altfel, din punct de vedere al performanței, selectarea unei constante asigură mai mare rapiditate decât selectarea unei coloane.

Se presupune că se dorește determinarea producătorilor de laptopuri, care produc și imprimante.



Producătorii de imprimante sunt preluați de subinterogare și comparați cu producătorul returnat din interogarea principală. Interogarea principală returnează producătorii de laptopuri. Astfel, pentru fiecare producător de laptopuri, se verifică faptul că subinterogarea returnează fiecare rând (prin urmare, acest producător produce, de asemenea, și imprimante). Deoarece ambele interogări din clauza *WHERE* trebuie simultan să fie satisfăcute (*AND*), rezultatul scontat va include doar liniile dorite. Cuvântul-cheie *DISTINCT* este folosit pentru a garanta că fiecare producător este returnat doar o singură dată.

Un avantaj al operatorului *EXISTS* este dat de faptul că nu mai este continuată căutarea în cererea internă după ce aceasta regăsește o linie. Ca alternativă a lui *EXISTS*, poate fi utilizat operatorul *IN* într-o cerere nesincronizată.

4.5.7. Subinterogări în clauza FROM

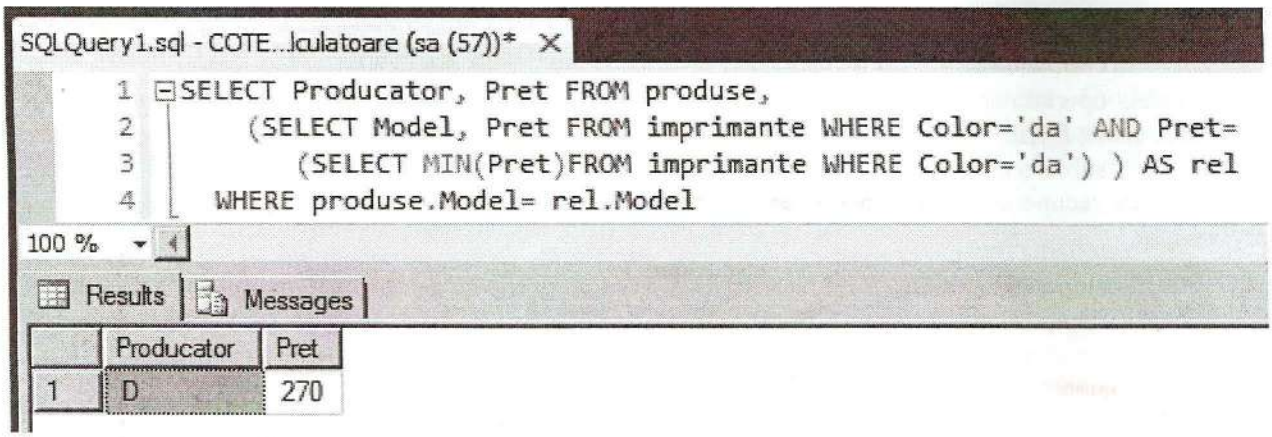
Până acum, în clauza *FROM*, a fost specificată o listă de tabele. Deoarece rezultatul unei instrucțiuni *SELECT* este un tabel, apare întrebarea dacă o interogare poate participa după *FROM* în calitate de tabel. Răspunsul este afirmativ. Formatul unei subinterogări în clauza *FROM* arată precum urmează:

```
...(<instrucțiune SELECT>) AS <alias> [(<coloana1>[,<coloana2> ...])]...
```

În acest format, `<alias>` este aliasul care se atribuie tabelului construită de subinterogare, ale cărei coloane, de asemenea, pot fi opțional atribuite nume noi. Numărul de coloane ale aliasului coincide cu numărul de coloane returnat de subinterogare. La rândul său, subinterogarea, evident, poate conține subinterogări (subinterogări simple fără operații pe mulțimi).

Aceste expresii pot fi specificate într-o listă a clauzei *FROM* exact ca un tabel. Interogarea principală poate face referire la coloanele subinterogării ca la coloanele oricărui tabel obișnuit.

Următoarea interogare cere să se găsească producătorii celor mai ieftine imprimante color și să se afișeze câmpurile ***Producator*** și ***Pret***. Un *SELECT* imbricat stă după clauza *FROM* și se utilizează de *SELECT*-ul principal ca un tabel virtual numit ***rel***. Acest *SELECT* imbricat, la rândul său, mai conține o subselecție imbricată în calitate de filtru.



4.5.8. Interogări cu operatori din teoria mulțimilor

Operatorii pe mulțimi combină rezultatele obținute din două sau mai multe interogări. Cererile care conțin operatori pe mulțimi se numesc cereri compuse.

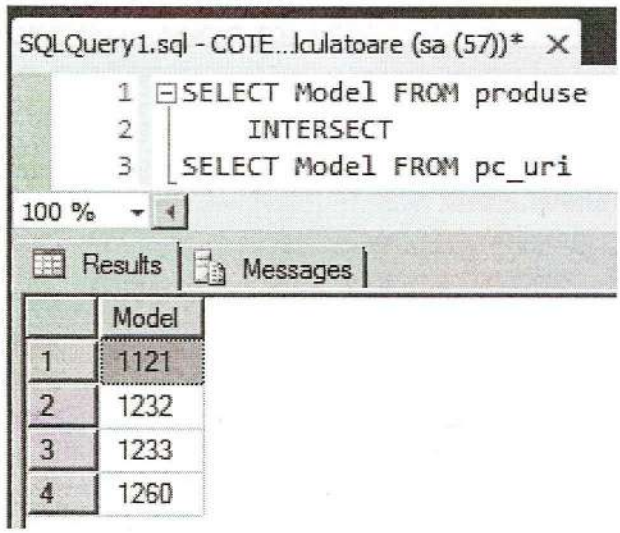
Interogările cu operațiile uniunea, intersecția și diferența sunt foarte aproape de interogările scrise în algebra relațională. Se formulează două interogări care construiesc tabele cu scheme compatibile, adică cu același număr de câmpuri și de același tip, și aceste cereri sunt legate cu cuvintele-cheie. Există patru operatori pe mulțimi: *UNION*, *UNION ALL*, *INTERSECT* sau *EXCEPT*.

Sintaxa pentru utilizarea operatorilor din teoria mulțimilor este:

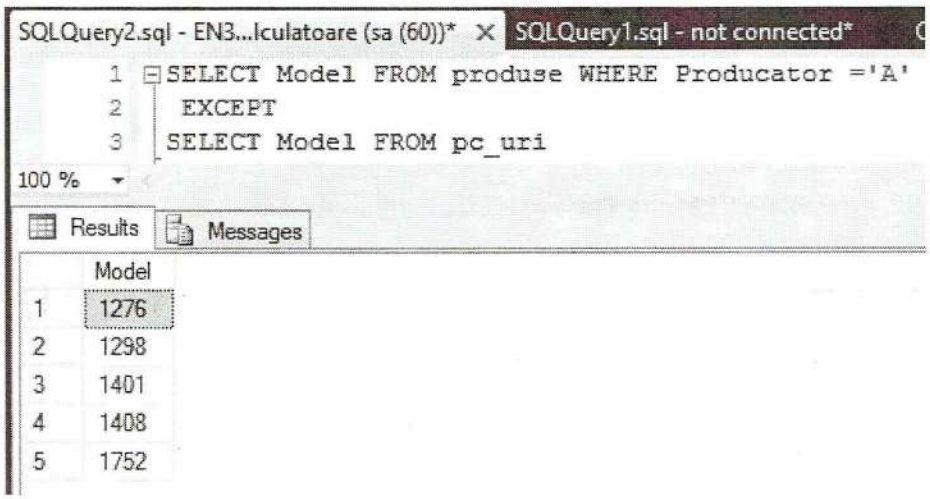
```
<interogare SQL>
    {UNION [ALL]| INTERSECT | EXCEPT}
<interogare SQL> [...]
```

Operatorii din teoria mulțimilor presupun eliminarea duplicatelor ca opțiune implicită. Dacă se dorește utilizarea acestor operatori cu menținerea duplicatelor, este suficientă specificarea opțiunii *ALL*. Opțiunea *ALL* poate conserva duplicatele doar cu utilizarea operatorului *UNION ALL*.

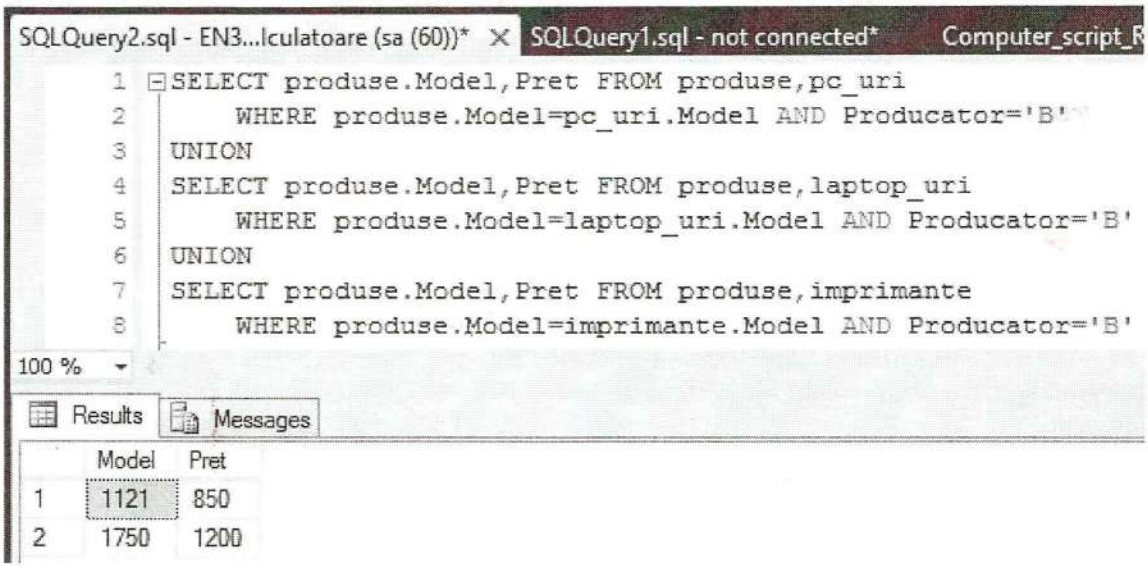
Pentru a găsi valori ce coincid în două sau mai multe tabele, se poate utiliza instrucțiunea *INTERSECT*, ca în exemplul de mai jos:



Dacă în exemplul de mai sus, operatorul *INTERSECT* este substituit cu *EXCEPT*, se obține un rezultat, într-un fel, invers celui precedent, adică se vor afișa modele de produse din primul tabel, care nu sunt în al doilea tabel.



Operația *UNION* permite fuzionarea a două selecții pentru a obține o mulțime de rânduri egală cu uniunea acestor două selecții. Un exemplu poate fi următoarea interogare: să se găsească modelele şi prețurile tuturor produselor (de orice tip) fabricate de compania 'B'.



4.6. Structuri de control ale fluxului

SQL este un limbaj neprocedural cu format liber, deoarece precizează ce rezultate sunt necesare, şi nu procedura prin care se obțin acestea. Implementarea limbajului Transact SQL permite realizarea de prelucrări complexe de tip procedural prin intermediul instrucțiunilor ce controlează fluxul execuției în pachetele de comenzi.

Instrucțiunile de limbaj pentru controlul fluxului în Transact SQL facilitează extinderea structurilor cu cele alternative, repetitive, selective şi condiționale complexe.

Cele mai importante instrucțiuni, din aceste categorii, sunt: *BEGIN...END, IF...ELSE, WHILE, CASE, BREAK, CONTINUE, RETURN, GOTO, WAITFOR* etc.

4.6.1. Blocuri Transact-SQL

Orice unitate Transact-SQL conține unul sau mai multe blocuri, complet separate sau imbricate.

O instrucțiune compusă se numeşte bloc. Un bloc permite tratare mai multe instrucțiuni ca pe una singură. Cu alte cuvinte, instrucțiunea compusă sau blocul sunt echivalente sintactic cu o singură instrucțiune. Un bloc Transact-SQL este compus din până la 3 secțiuni: declarativă (opțională), executabilă (obligatorie) şi de tratare a excepțiilor (opțională).

Există 3 tipuri de blocuri:

- **Blocuri anonime:** acestea sunt blocuri Transact-SQL, care sunt definite în cadrul unei aplicații și nu au nume.
- **Proceduri:** acestea sunt blocuri Transact-SQL, care au un nume, au parametri de intrare, dar nu au parametri de ieșire expliciți.
- **Funcții:** acestea sunt blocuri Transact-SQL, care au un nume, au parametri de intrare și tot timpul returnează o valoare.

Corpul unei funcții este o instrucțiune compusă. Domeniul de vizibilitate al unui identificator declarat într-un bloc se întinde din punctul declarației până la sfârșitul blocului. Identificatorii utilizați într-un bloc pot fi ascunși prin declarații de același nume în blocurile interioare ale blocului inițial.

La rândul lor:

- **Pachete de programe:** grupează proceduri, funcții.
- **Declanșatoare pe bază de date:** blocuri Transact-SQL asociate tabelelor (de bază sau virtuale) și lansate automat în execuție când are loc o comandă de manipulare.
- **Declanșatoare de aplicație:** blocuri Transact-SQL asociate unor evenimente din cadrul aplicației și lansate în execuție automat.

Blocul compus dintr-un cod executabil (obligatoriu): această secțiune începe cu cuvântul-cheie *BEGIN* și se sfârșește cu *END*.

Sintaxa unui bloc de instrucțiuni are structura:

```
BEGIN
      {
        <instrucțiune_sql> | <bloc instrucțiuni sql>
      }
END
```

unde { <instrucțiune_sql> | <bloc instrucțiuni sql> } reprezintă orice instrucțiune sau grup de instrucțiuni valide definite într-un bloc de instrucțiuni. Blocurile *BEGIN ... END* pot fi imbricate.

Deși toate declarațiile Transact-SQL pot fi valabile într-un bloc *BEGIN ... END*, unele dintre ele nu ar trebui să fie grupate într-un singur pachet (bloc de instrucțiuni).

4.6.2. Inițializarea variabilelor locale

Un bloc Transact-SQL poate fi construit și din alte secțiuni, precum cea declarativă. Secțiunea declarativă începe cu instrucțiunea *DECLARE* și este opțională; ea poate conține o serie de declarații, inclusiv a variabilelor.

În limbajul Transact-SQL pot fi folosite variabile locale pentru memorarea unor valori care pot fi testate sau modificate, în plus, asigură transferul datelor către și de la tabelele bazei de date. Variabilele locale au ca domeniu de definiție lotul, procedura sau blocul în care au fost declarate.

O variabilă locală se declară folosind instrucțiunea *DECLARE* care specifică un identificator (un nume care trebuie să înceapă cu caracterul @) și tipul variabilei.

Sintaxa de declarare arată astfel:

```
DECLARE @<nume_variabila>  <tip_date>
```

Limbajul Transact-SQL suportă toate tipurile de date prevăzute în standardul SQL2, în plus, permite definirea de către utilizator a unor noi tipuri de date.

Într-o linie de cod Transact-SQL, pot fi plasate mai multe declarații de variabile (în calitate de alternativă fiecare declarație poate fi plasată pe o linie separată). De exemplu:

```
DECLARE @MyChar CHAR(9), @MyINT INTEGER, @MyDate DATETIME
```

Plasarea declarațiilor fiecărei variabile pe o linie separată elimină necesitatea utilizării virgulei, dar, din păcate, suntem obligați, la începutul fiecărei linii, să precizăm cuvântul-cheie *DECLARE*.

```
DECLARE @MyChar CHAR(9)
DECLARE @MyINT INTEGER
DECLARE @MyData DATETIME
```

De regulă, cea mai importantă sarcină, după declarație, este cea de inițializare a variabilelor. Inițializarea – atribuirea valorii inițiale unei variabile. Una dintre recomandările Microsoft – pentru atribuirea valorii unei variabile să se folosească operatorul *SET*, în loc de instrucțiunea *SELECT*. În continuare, urmează exemple de utilizare a doi operatori, cu toate acestea, cum menționează Microsoft, operatorul *SET* este special optimizat pentru a efectua operațiile de atribuire.

```
DECLARE @MyChar CHAR(9)
SET @MyChar = 'SQLServer'
```

Alternativ, în loc de operatorul *SET*, poate fi folosită instrucțiunea *SELECT*:

```
DECLARE @MyChar CHAR(9)
SELECT @MyChar = 'SQLServer'
```

Limbajul Transact-SQL suportă toate instrucțiunile SQL, cu unele modificări de sintaxă, astfel încât să poată fi folosite în combinație cu variabilele locale ale programului. De exemplu, forma modificată a instrucțiunii *SELECT*, prin care se asignează variabile locale cu valori ale unor atribute selectate, este:

```
SELECT @<var1> = <expresie1> [,@<var2> = <expresie2> ...]
    FROM <lista_tabele> [WHERE <condiție>]
```

O astfel de instrucțiune este utilă pentru interogările care returnează o singură linie, deoarece variabilele locale sunt setate cu valorile coloanelor primei linii a rezultatului, iar valorile din celelalte linii se pierd (nemaifiind loc unde să fie memorate). Atunci, când o interogare returnează o mulțime de linii, se poate folosi un cursor.

4.6.3. Structura selectivă CASE

Expresia *CASE* este o adăugire recentă, dar foarte importantă, la limbajul SQL. Pentru prima dată, părți ale instrucțiunilor SQL pot fi executate condițional. De exemplu, o coloană din rezultatele interogării poate fi formatată în funcție de valorile conținute pe o altă coloana.

Expresia *CASE* are următoarea sintaxă generală:

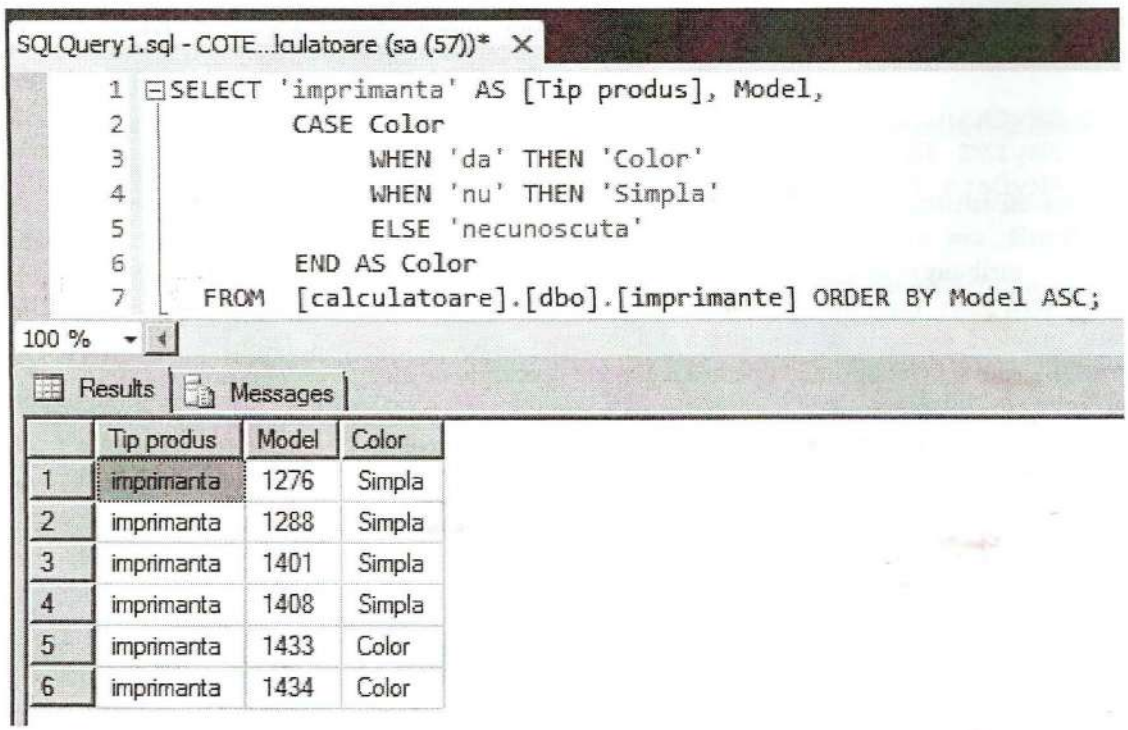
```
CASE <expresie_intrare>
     WHEN <expresie_comparare> THEN <expresie_rezultat>
    [WHEN <expresie_comparare> THEN <expresie_rezultat>...]
    [ELSE <expresie_rezultat_alternativ>]
END
```

Fiecare condiție *WHEN* este evaluată sub forma `<expresie_intrare>` = `<expresie_intrare>` şi, dacă rezultatul are valoarea *TRUE*, este returnată valoarea `<expresie_rezultat>` şi nici o altă condiție *WHEN* nu mai este evaluată.

Dacă nici una dintre condițiile *WHEN* nu este evaluată ca adevărată şi exista o condiție *ELSE*, este returnată valoarea `<expresie_rezultat>` asociată condiției *ELSE*.

Dacă nici una dintre condițiile *WHEN* nu este evaluată ca adevărată şi nu există o condiție *ELSE*, este returnată valoarea *NULL*.

În exemplul din imaginea de mai jos este evaluat cu ajutorul expresiei *CASE* câmpul ***Color*** din tabelul ***imprimante***. În funcție de valoarea câmpului, este returnat un text mai informativ pentru utilizatorul bazei de date.



CASE poate fi folosit în orice instrucțiune sau clauză care permite o expresie validă. De exemplu, *CASE* poate fi utilizat in instrucțiunile *SELECT*, *UPDATE*, *DELETE* şi *SET*, precum şi în clauzele *IN*, *WHERE*, *ORDER BY*, *HAVING* şi în lista după clauza *SELECT*.

4.6.4. Structura alternativă IF...ELSE

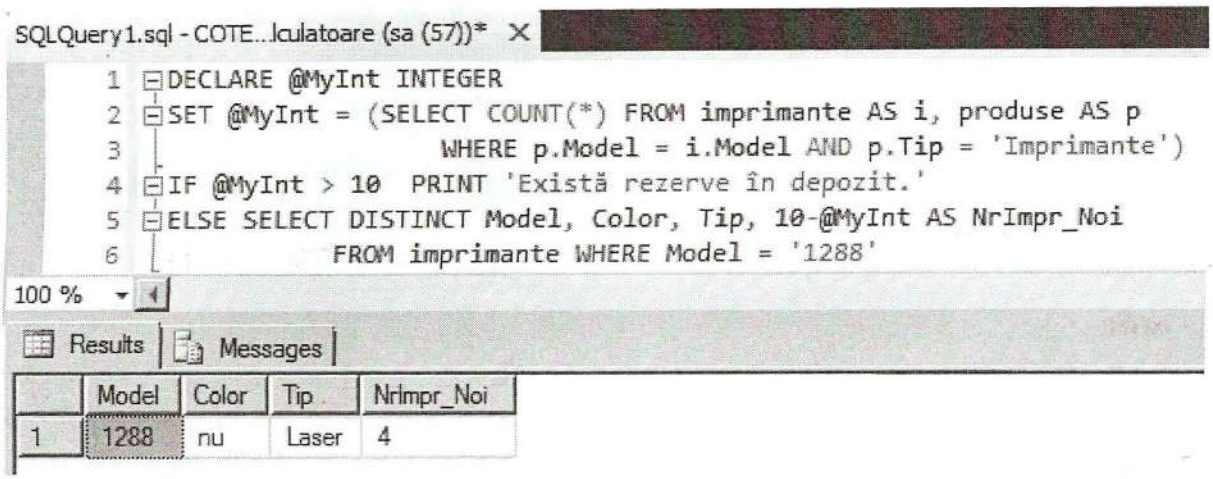
Structura de control alternativ modifică fluxul de lucru, executând sau nu anumite porțiuni de cod, în funcție de îndeplinirea sau neîndeplinirea unor condiții. Probabil, cea mai des utilizată structură de control al fluxului este *IF...ELSE*.

Instrucțiunea *IF...ELSE*, cu care, în Transact-SQL se implementează o structură alternativă de luare a deciziilor, are următorul format general:

```
IF <expresie_booleană>
      { <instrucțiune_sql1> | <bloc instrucțiuni sql1> }
  [ ELSE
    { <instrucțiune_sql2> | <bloc instrucțiuni sql2> } ]
```

Instrucțiunea *IF...ELSE* impune următoarele condiții de execuție: Dacă valoarea `<expresie_booleană>` este *TRUE*, se execută instrucțiunea sau blocul de instrucțiuni `{<instrucțiune_sql1> | <bloc instrucțiuni sql1>}`. Clauza *ELSE* este opțională şi ea este urmată de instrucțiunile, care trebuie să fie executate, dacă valoarea expresiei `<expresie_booleană>` este *FALSE* sau *NULL*.

În următorul exemplu, sunt utilizate cuvintele-cheie *IF* şi *ELSE*, şi se verifică condițiile de executare a acestei structuri. Dacă numărul de imprimante produse este mai mare de 10, se execută instrucțiunea de tipărire a faptului că, în depozit, sunt rezervele necesare. În caz contrar, se eliberează un număr necesar de imprimante de ultimul model 1288.



Să se observe în exemplul dat modul de utilizare a variabilelor locale, care sunt valabile în cadrul acestei instrucțiuni.

4.6.5. Structura repetitivă WHILE

Structurile de control repetitive execută anumite porțiuni de cod de un număr determinat de ori. În sistemul Transact-SQL, există o posibilitate – aplicarea operatorului *WHILE*. Operatorul *WHILE* testează o condiție booleană şi execută repetat o instrucțiune sau un bloc de instrucțiuni. Executarea acestor instrucțiuni se va repeta atâta timp cât condiția specificată va rămâne adevărată.

Corpul instrucțiunii poate fi o instrucțiune vidă, o instrucțiune simplă sau un bloc de instrucțiuni ce vor fi executate repetitiv. Deoarece instrucțiunea *WHILE* realizează testarea condiției la începutul instrucțiunii, această instrucțiune este bună de utilizat în situațiile în care nu se doreşte executarea buclei, evident, dacă condiția nu este adevărată.

Sintaxa, în forma sa generală, a operatorului, arată după cum urmează:

```
WHILE <expresie_booleană>
{ <instrucțiune_sql> | <bloc instrucțiuni sql> | BREAK | CONTINUE }
```

`<expresie_booleană>`

Evaluează expresie şi returnează *TRUE* sau *FALSE*. În cazul în care expresia booleană conține o instrucțiune *SELECT*, aceasta se va scrie între paranteze rotunde.

`{<instrucțiune_sql> | <bloc instrucțiuni sql> }`

Reprezintă orice instrucțiune sau grup de instrucțiuni, definite în formă de bloc de instrucțiuni. Un bloc de instrucțiuni este o secvență, eventual vidă, de instrucțiuni. Aceste instrucțiuni se execută în ordinea în care apar în interiorul blocului. Sintactic, blocul de instrucțiuni începe cu cuvântul-cheie *BEGIN* şi se termină cu cuvântul-cheie *END*.

`BREAK`

Se realizează ieşirea forțată din *WHILE* Toate instrucțiunile situate după cuvântul-cheie *END*, care marchează sfârşitul buclei *WHILE*, sunt executate.

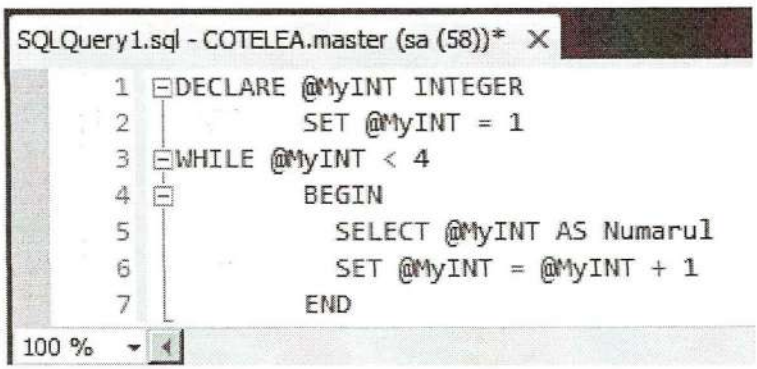
`CONTINUE`

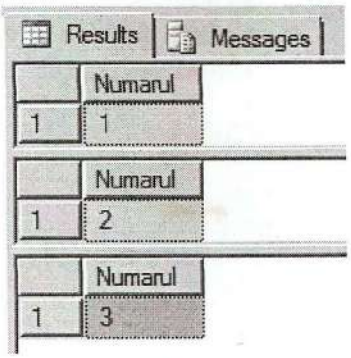
Cauzează repornirea buclei *WHILE*, ignorând toate instrucțiunile care urmează după cuvântul *CONTINUE*.

În timpul executării, se evaluează mai întâi condiția buclei a cărei valoare trebuie să fie întreagă. Dacă valoarea calculată este *TRUE*, atunci instrucțiunea se execută. Când condiția, adică valoarea (inclusiv prima) a expresiei booleene din paranteze este *FALSE*, se face un salt la instrucțiunea de după corpul instrucțiunii *WHILE*, deci, instrucțiunea *WHILE* se termină. Astfel, instrucțiunea asociată cu *WHILE* se execută repetat, cât timp valoarea asociată condiției este diferită de *FALSE*.

Aşadar, operatorul *WHILE* se execută de zero sau mai multe ori. Executarea instrucțiunilor buclei *WHILE* poate fi controlată din interiorul buclei cu ajutorul cuvintelor-cheie *BREAK* şi *CONTINUE*.

Următorul exemplu utilizează bucla *WHILE*, afişând valorile unui contor:





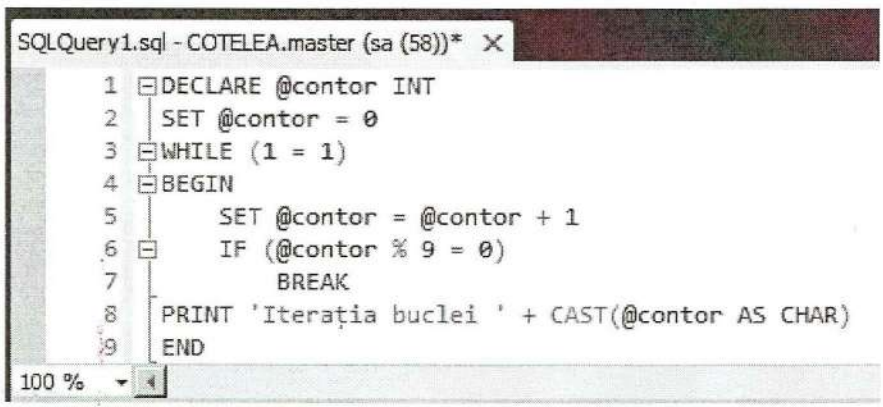
MS SQL Server va executa repetat bucla atâta timp cât valoarea nu va deveni egală cu 4. Testul buclei va avea loc înaintea fiecărei execuții a corpului.

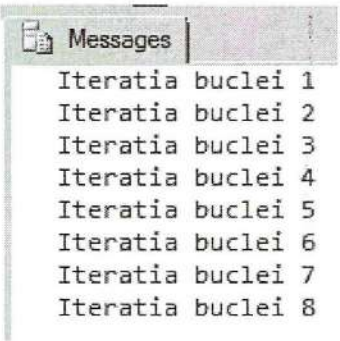
4.6.6. Instrucțiunea BREAK

Adesea este convenabil să se controleze ieşirile din bucle altfel decât testând condiția la începutul sau sfârşitul buclei. Instrucțiunea *BREAK* oferă o ieşire mai devreme din instrucțiuni. *BREAK* permite ieşirea dintr-o buclă, din oricare punct din interiorul instrucțiunii *WHILE* sau al instrucțiunilor *IF ... ELSE* sau *CASE*. Când instrucțiunea *BREAK* este întâlnită în interiorul unei bucle, aceasta se termină imediat. Se parcurge, în continuare, instrucțiunea care urmează după buclă.

O buclă din care se poate ieşi (folosind *BREAK*), după un număr cunoscut de paşi sau la îndeplinirea unei condiții (ieşire forțată), este, de obicei, urmată de o instrucțiune *IF...ELSE* care stabileşte cum s-a ieşit din buclă: fie după numărul maxim de paşi, fie mai înainte datorită satisfacerii condiției. Instrucțiunea *BREAK* poate fi utilizată cu oricare dintre celelalte instrucțiuni ale limbajului Transact SQL, această instrucțiune producând ieşirea.

În exemplul următor, repetarea buclei va fi întreruptă de instrucțiunea *BREAK*:





Utilizarea instrucțiunii *BREAK* poate simplifica expresiile din *WHILE* şi poate contribui la urmărirea mai uşoară a codului, deşi se poate evita instrucțiunea *BREAK* prin complicarea expresiei testate în *WHILE*. În general, se recomandă ca instrucțiunea *BREAK* să fie utilizată pentru scopuri speciale, şi nu ca ieşire normală din buclă.

4.6.7. Instrucțiunea CONTINUE

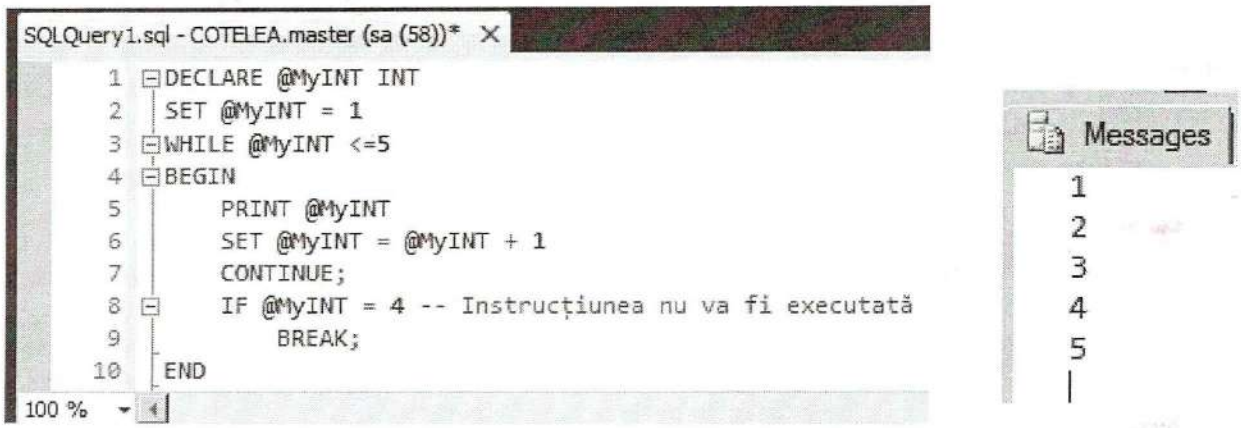
Instrucțiunea *CONTINUE* este opusă instrucțiunii *BREAK*. Nu se iese în afara buclei, ca în cazul instrucțiunii *BREAK*. Ea face să înceapă următoarea iterație a buclei *WHILE*. Aceasta înseamnă că partea de test se execută imediat.

Efectul instrucțiunii *CONTINUE* constă în oprirea iterației curente a buclei şi un salt imediat la prima instrucțiune din buclă, pentru a continua cu următoarea iterație. Orice instrucțiune după cuvântul-

cheie *CONTINUE* este ignorată. Cu alte cuvinte, ea forțează următoarea iterație a buclei să aibă loc, trecând peste instrucțiunile dintre ea și testul de condiție.

CONTINUE se aplică, de regulă, numai la bucle, nu și la *CASE*. Un *CONTINUE* înăuntrul unui *CASE* dintr-o buclă declanșează următoarea iterație a buclei. Deși acest lucru nu este o regulă absolută, *CONTINUE* este adesea deschis printr-un test *IF...ELSE*.

În exemplu, execuția buclei (ciclului *WHILE*) folosește cuvintele-cheie *CONTINUE* și *BREAK*. Instrucțiunea *BREAK* nu va fi executată.



Instrucțiunea *CONTINUE* este folosită rareori, nu pentru că folosirea ei este incorectă, ci pentru că aplicațiile în care ar putea fi utilizată sunt mai rare.

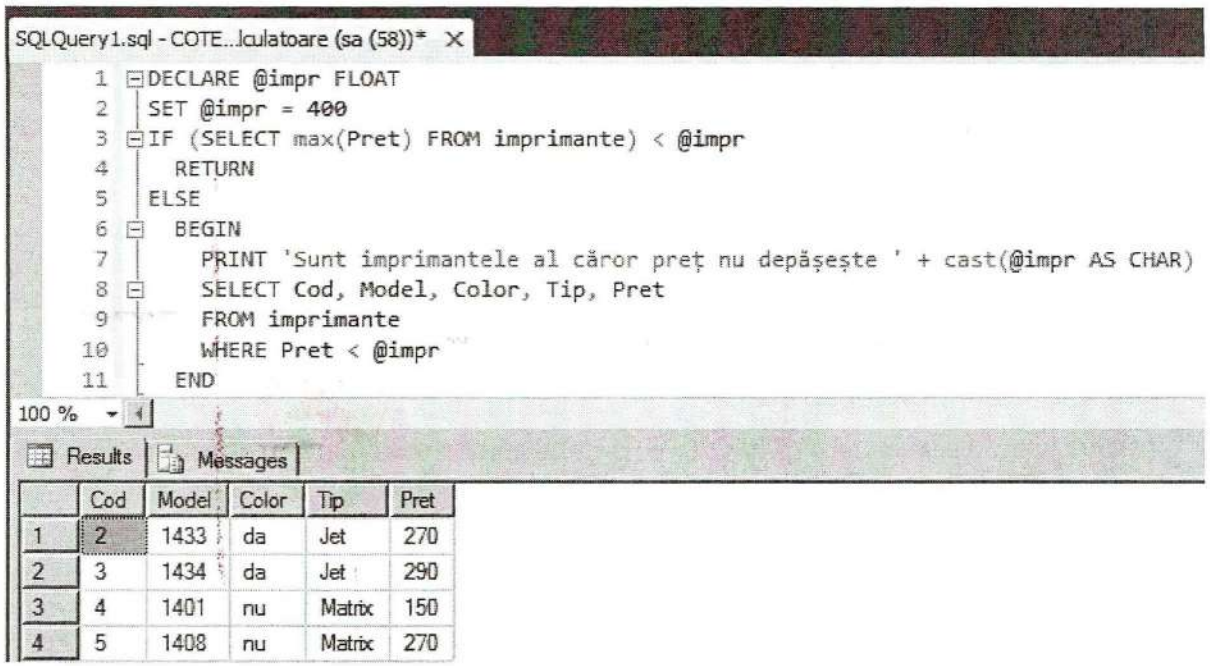
4.6.8. Instrucțiunea RETURN

Uneori, este nevoie de o abandonare prematură a procesului de executare a instrucțiunilor. Această necesitate apare, de exemplu, în cazul când se verifică anumiți parametri și parametrii dau rezultatul greșit. Prin intermediul comenzii *RETURN*, se poate forța întreruperea unui proces. Comenzile ce urmează după instrucțiunea *RETURN* nu vor mai fi executate.

Sintaxa instrucțiunii *RETURN* este:

```
RETURN [<expresie de tip întreg> ]
```

Următorul exemplu ilustrează utilizarea Transact-SQL cu instrucțiunea *RETURN:*



	Cod	Model	Color	Tip	Pret
1	2	1433	da	Jet	270
2	3	1434	da	Jet	290
3	4	1401	nu	Matrix	150
4	5	1408	nu	Matrix	270

RETURN permite părăsirea necondiționată a unei interogări sau proceduri. *RETURN* are o acțiune imediată și completă, și poate fi utilizabilă în orice punct, pentru a părăsi o procedură, pachet sau bloc de instrucțiuni. Instrucțiunile următoare, după *RETURN,* nu sunt executate.

După cum se poate observa, opțional, după instrucțiunea *RETURN,* se poate preciza un număr întreg ce poate fi utilizat ulterior în cadrul blocului de instrucțiuni ce a lansat în execuție procedura.

4.6.9. Goto-uri și etichete

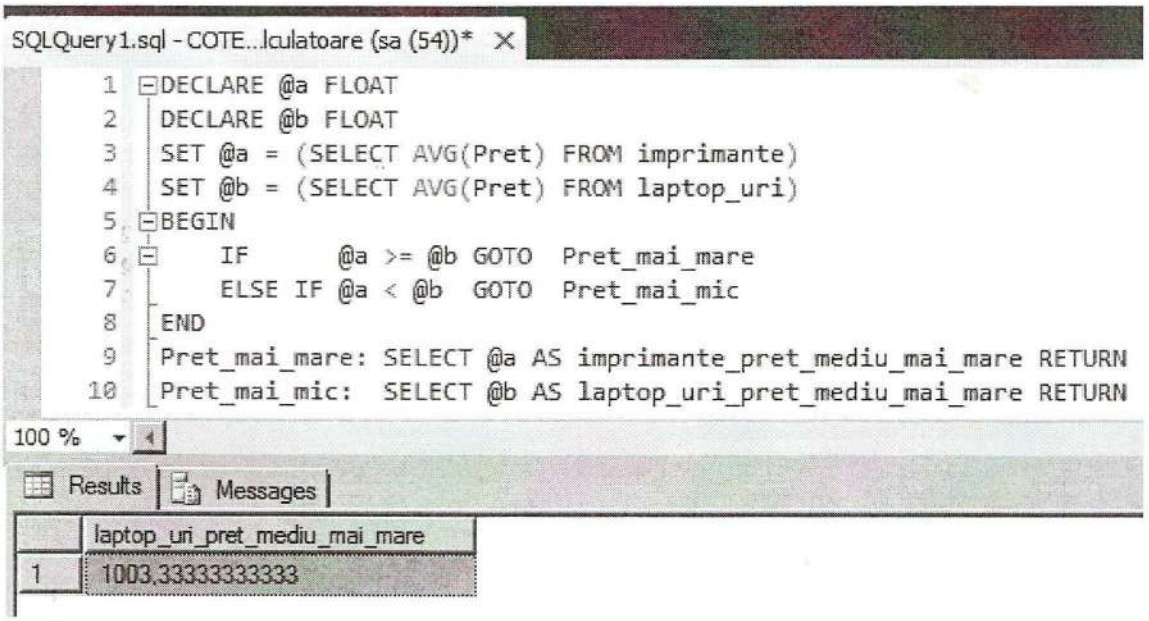
Instrucțiunea *GOTO* oferă controlul selectiv al acțiunilor și se folosește la saltul necondiționat la o etichetă. Instrucțiune care urmează după *GOTO* este ignorată și procesarea continuă la etichetă. Instrucțiunile *GOTO* și etichetele pot fi folosite, în orice loc, al unei proceduri, pachet sau bloc de instrucțiuni.

Sintaxa instrucțiunii:

```
GOTO <etichetă>
```

Instrucțiunile *GOTO* pot fi imbricate. Formal, *GOTO* nu este necesară niciodată și, în practică, este aproape întotdeauna ușor să fie scris un cod fără ea. Cu toate acestea, poate fi sugerată o situație în care *GOTO* își poate găsi locul. Cea mai obișnuită folosire este aceea de a abandona prelucrarea în anumite structuri puternic imbricate, de exemplu, de a ieși afară din două bucle deodată. Instrucțiunea *BREAK* nu poate fi folosită, deoarece ea părăsește numai bucla cea mai din interior.

În exemplu, se calculează prețul mediu cel mai mare dintre ***imprimante*** și ***laptop_uri:***



Codul, implicând un *GOTO,* poate fi scris întotdeauna fără *GOTO,* chiar dacă prețul pentru aceasta este o variabilă suplimentară, sau teste repetate. Cu toate că, în privința subiectului, instrucțiunea *GOTO* muta controlul execuției către una etichetată, codul este puțin lizibil și se pare că instrucțiunea *GOTO* ar trebui folosită cu economie (codul este puțin lizibil), dacă nu chiar deloc.

4.6.10. Operatorului WAITFOR

Cu ajutorul operatorului *WAITFOR,* interogările pot fi trecute în regimul de așteptare. Urmează două exemple de utilizare a "așteptării".

Se dorește compararea datelor dintr-un tabel modificate peste o oră. Pentru a face acest lucru, se selectează datele necesare într-un tabel temporar, iar peste o oră se compară conținutul tabelului nou cu datele salvate anterior. Dacă nu ar exista posibilitatea de utilizare a instrucțiunii *WAITFOR,* vor trebui aplicate două scenarii separate, cu o pauză de o oră între ele. Dar, datorită operatorului *WAITFOR,* toate acestea se poate face în mod automat, fără costuri suplimentare de muncă.

Un alt exemplu de aplicare ar fi cazul când se dorește fixarea unui scenariu specific, de exemplu, la sfârșitul zilei de lucru. Dacă aceasta ar fi fost o sarcină zilnică, se poate trage concluzia că activitatea manuală ar fi foarte neeficientă.

În general, operatorul arată, precum urmează:

```
WAITFOR
{
    DELAY 'timp așteptare'
  | TIME 'ora execuție'
}
```

`DELAY`

Perioada de timp, care trebuie să treacă (nu mai mult de 24 de ore) până la continuarea executării unui pachet, proceduri stocate sau tranzacții.

`'timp așteptare'`

Argumentul „timpul de așteptare" poate fi specificat în orice format acceptat de date *DATETIME*, sau ca o variabilă locală. Dacă nu pot fi specificate datele, partea valorii *DATETIME* nu este admisă.

`TIME`

Specifică timpul de executare a pachetului, procedurii stocate sau tranzacției.

`'ora execuție'`

Timpul la care instrucțiunea *WAITFOR* se termină. Argumentul poate fi redat în orice format acceptat de date *DATETIME* sau de o variabilă locală. Dacă datele nu pot fi specificate, prin urmare, o parte a valorii *DATETIME* nu va fi autorizată.

Astfel, operatorul *WAITFOR* stabilește un timp de întârziere 'timp așteptare' până la momentul lansării în execuție a unei instrucțiuni SQL sau stabilește ora ('ora execuție'), la care se va lansa în execuție o instrucțiune SQL.

În următorul exemplu, este prezentat modul în care se creează o întârziere pe un minut. Interogarea *SELECT* realizează preluarea datelor peste un minut:

```
WAITFOR DELAY '00:01'
      SELECT * FROM produse
```

Preluarea datelor din tabelul ***produse*** întârzie până la ora 18:30. Instrucțiunea e redată în următorul exemplu:

```
WAITFOR TIME '18:30'
      SELECT * FROM produse
```

Întrebări de control:

1. Ce facilități oferă componenta SQL Transact-SQL Query Editor ?
2. Ce reprezintă DDL, DML, DCL și TCL?
3. Ce operatori oferă limbajul Transact SQL?
4. Care este sintaxa de bază a instrucțiunii *SELECT*?
5. Ce funcții oferă limbajul Transact SQL (inclusiv funcții de agregare)?
6. Interogări cu *JOIN*. Tipuri de joncțiuni.
7. La ce servesc interogările corelate?
8. Cum pot fi redate operațiile *INTERSECT* și *EXCEPT* dintre două subinterogări prin interogări imbricate?
9. Tipuri de blocuri în Transact-SQL.

10. Să se explice funcționarea buclei repetitive *WHILE*.
11. Să se explice funcționarea expresiei *CASE*.
12 Care sunt metodele de limitare a numărului de linii returnate de interogare?

Sarcini practice:

Sarcinile vor fi aplicate asupra bazei de date (care conține date despre calculatoare), a cărei descriere scurtă este propusă în continuare.

Schema bazei de date constă din patru scheme relaționale:

produse(Producator,Model, Tip)
pc_uri(Cod, Model, Viteza, Ram, Hd, Cd, Pret)
laptop_uri(Cod, Model, Viteza, Ram, Hd, Ecran, Pret)
imprimante(Cod, Model, Color, Tip, Pret)

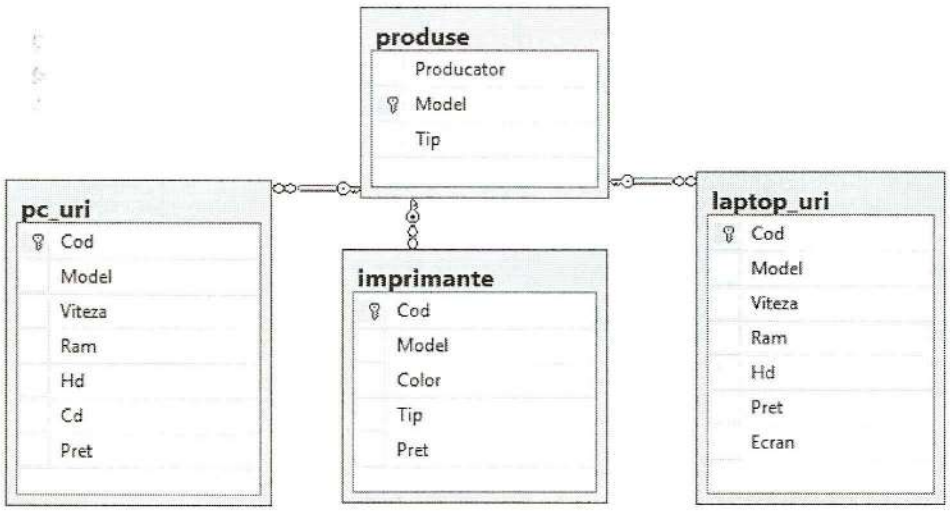
Tabelul ***produse*** conține date despre producător (***Producator***), numărul modelului (***Model***) și tipul (*pc, laptop, imprimante*) produsului. Se presupune că numărul modelului este unic pentru fiecare producător și tip de produs.

În ceea ce privește tabelul ***pc_uri***, pentru fiecare număr de model este indicată viteza procesorului în megahertzi (***Viteza***), capacitatea memoriei ***Ram*** (în Mb), capacitatea discului ***Hd*** (în Gb), viteza dispozitivului de citire ***Cd*** (de exemplu, 4x) și prețul – ***Pret***.

Tabelul ***laptop_uri*** este asemănătoare tabelului ***pc_uri***, cu excepția că, în locul vitezei *Cd*-ului, este prezentă dimensiunea ecranului – ***Ecran*** (în inch).

În tabelul ***imprimante***, pentru fiecare model, este indicat faptul dacă ea este color sau nu – ***Color*** ('*da*' - dacă color) și, de asemenea, tipul imprimantei – ***Tip*** (*Laser, Jet* sau *Matrix*) și prețul.

În figura ce urmează, este reprezentată diagrama bazei de date:



Cele mai simple interogări

1. Să se găsească modelul, viteza procesorului și capacitatea discului dur pentru toate *pc*-urile care costă mai puțin de 500$. Schema rezultatului afișat va fi ***Model***, ***Viteza*** și ***Hd***.
2. Să se găsească producătorii de imprimante. Afișați: ***Producator***
3. Să se găsească modelul, capacitatea de memorie operativă și dimensiunea ecranului fiecărui *laptop* cu prețul mai mare de 1000$.
4. Să se găsească toate liniile tabelului ***imprimante*** pentru imprimantele color.
5. Să se găsească modelul și viteza procesorului, și capacitatea discului dur ale *pc*-urilor mai ieftine de 600$ și care posedă *cd-rom-uri* cu viteze 12x sau 24x
6. Să se afișeze producătorii și viteza *pc*-urilor ce posedă HD-uri cu capacitatea cea mai mică.

7. Să se găsească modelele şi producătorii produselor cu preţul mai mare de 450 dolari
8. Să se găsească producătorii care vând *laptopuri* sau *imprimante*, dar nu vând *pc*-uri.
9. Să se găsească producătorii de *pc*-uri cu procesoare ce funcţionează cu frecvenţa de tact nu mai mică de 450Mhz. Afişaţi câmpul ***Producător***.
10. Să se găsească imprimantele cu cel mai înalt preţ. Să se afişeze ***Model***, ***Pret***.
11. Să se găsească viteza medie (***Viteza***) a *pc*-urilor.
12. Să se găsească viteza medie a *laptopurilor*, al căror preţ depăşeşte 1000$.
13. Să se găsească viteza medie a *pc*-urilor produse de compania '*A*'.
14. Să se găsească modele şi producătorii *pc*-urilor care au o viteză mai mare decât cel mai rapid *laptop*.
15. Afişarea tuturor producătorilor *pc*-urilor ce au *Ram*-ul mai mare decât media *Ram*-ului laptopurilor. Să se afişeze producătorul şi modelul *pc*-urilor.
16. Să se găsească modelele laptopurilor cu preţul mai mare decât preţul dublu al celui mai ieftin *pc*. Să se afişeze modelul laptopului şi preţul acestuia
17. Pentru fiecare valoare a vitezei, să se găsească preţul mediu al *pc*-urilor. Să se afişeze ***Viteza*** şi ***Pret_mediu***.

Interogări multi-relaţie

1. Să se găsească capacităţile discurilor dure, care coincid cu 2 sau mai multe *pc*-uri. Să se afişeze ***Hd***.
2. Să se găsească perechile de modele de *pc*_uri, care posedă procesoare cu aceeaşi viteză şi memorie *Ram* cu aceeaşi capacitate. Ca rezultat, fiecare pereche se afişează o singură dată, adică (i,j), dar nu (j,i) Ordinea de afişare: modelul cu număr mai mare, modelul cu număr mai mic, viteza şi *Ram*.
3. Să se găsească *laptopurile*, a căror viteză este mai mică decât la orice *pc-u*. Să se afişeze: ***Tip***, ***Model***, ***Viteza***.
4. Să se găsească producătorii celor mai scumpe *laptopuri* sau *pc-uri*. Afişaţi ***Producator***, ***Model*** şi ***Preţ***.
5. Să se găsească dimensiunea medie a ecranului *laptopurilor* produse de fiecare producător. Să se afişeze ***Producator*** şi dimensiunea medie a ecranului.
6. Să se găsească producătorii, care produc cel puţin 3 *pc*-uri. Afişaţi ***Producator***, numărul de modele.
7. Să se găsească preţul maximal al *pc*-urilor propuse de fiecare producător. Afişaţi ***Producator***, preţ maxim.
8. Pentru fiecare valoare a vitezei *pc*-ului care depăşeşte 600 Mh, să se determine preţul mediu al calculatorului cu aceeaşi viteză. Afişaţi ***Viteza***, preţ mediu.
9. Afişarea tuturor producătorilor ce produc *pc*-uri şi *laptopuri* care dispun de o viteză nu mai mică de 750Mh. Afişaţi ***Producator***.
10 Să se găsească suma totală a preţurilor celor mai ieftine imprimante de fiecare tip.
11. Să se găsească modelul produsului cu cel mai mare preţ. Se afişează ***Model***.
12. Să se găsească producătorii de imprimante care produc şi *pc*-uri cu cel mai mic volum RAM. Să se afişeze ***Producator***.
13. Să se găsească preţul mediu al *pc*-urilor şi laptopurilor produse de către 'A'. Să se afişeze ***Pret_med_comun***.
14. Să se găsească preţul mediu al *pc*-urilor pentru fiecare din producători care produc şi imprimante. Afişaţi ***Producator***, ***Pret_med_prod***.
15. Să se găsească capacitatea medie a HD-urilor *pc*-urilor (o valoare pentru toate) ale acelor producători care produc şi imprimante. Afişaţi capacitatea medie HD.
16. Să se găsească producătorii care produc numai un singur tip de produs. Să se afişeze producătorul, tipul produsului, modelul şi preţul modelului respectiv.

17. Sa se analizeze prețurile laptop-urilor şi in funcție de valoare sa fie grupate in 3 clase. Daca este mai ieftin de 750 de dolari – *Clasa Econom*, daca prețul nu depăşeşte 1100 de dolari – *Clasa Business*, iar dacă prețul este mai mare de 1100 dolari – *Clasa Premium*. Să se afişeze tipul produsului, ***Model***, ***Pret*** şi ***Clasa Produsului***.
18. In tabelul ***produse*** să se găsească modelele la care prima cifră din model este o cifra impara, iar ultima – pară şi prima cifră este mai mică decât ultima. Să se afişeze modelul, tipul produsului, produsul primei şi ultimei cifre din numărul modelului.
19. Să se calculeze suma cifrelor numerelor fiecărui model din tabelul ***produse***. Să se afişeze modelul şi suma. De exemplu, daca modelul este 1121, atunci suma cifrelor din model este 5.
20. Din tabelul ***produse*** să se obțină un tabel cu coloanele ***Producator***, ***PC***, ***Imprimante***, ***Laptop_uri***. Pentru fiecare producător, în acest tabel, să se indice daca acesta produce sau nu (*da* sau *nu*) un anumit tip de produse. De exemplu, un rând din tabelul construit poate arăta: *A, nu, da, nu.*

5. CREAREA TABELELOR ŞI INDECŞILOR

În capitolul de față sunt tratate structura instrucțiunilor *CREATE*, *ALTER*, *INSERT*, *UPDATE*, *DELETE* utilizate pentru crearea şi gestiunea tabelelor. În afară de aceasta, sunt examinate aspectele de creare şi administrare a indecşilor, utilizând instrucțiuni Transact-SQL, precum şi în mediul grafic Microsoft SQL Server Management Studio [2].

Sistemul SQL Server poate gestiona, într-o bază de date, până la două miliarde de tabele. Fiecare tabel poate să conțină până la 1024 coloane, iar dimensiunea maximă, în octeți, pentru o înregistrare (linie) este de 8060. Aceste restricții sunt mai puțin restrictive pentru tabelele cu coloane de tip *VARCHAR*, *NVARCHAR*, *VARBINARY* sau *SQL_VARIANT*.

5.1. Crearea şi modificarea structurii tabelelor în Transact-SQL

Limbajul Transact-SQL, pentru crearea şi modificarea structurii tabelelor, oferă instrucțiunile *CREATE TABLE* şi *ALTER TABLE*, respectiv.

5.1.1. Instrucțiunea CREATE TABLE

Atunci când se creează un tabel, folosind instrucțiunile SQL, acesta se va găsi în baza de date curentă. Pentru crearea tabelelor prin cod se utilizează instrucțiunea *CREATE TABLE*.

Instrucțiunea *CREATE TABLE* defineşte coloanele tabelului, tipurile de date ale coloanelor şi diferite constrângeri pe care datele înregistrate (valori ale coloanelor) trebuie să le respecte pentru asigurarea integrității bazei de date. Sintaxa generală a acestei instrucțiuni este:

```
CREATE TABLE <nume_tabel> (
   <coloana1> <tip_data1> [DEFAULT <expr1>] [<constr_coloana1>]
 [,<coloana2> <tip_data2> [DEFAULT <expr2>] [<constr_coloana1>], ...]
  [<constrangeri_tabel>]  );
```

- `<nume_tabel>` - *indică numele tabelului creat;*
- `<coloana>` - *indică numele câmpului ce trebuie creat în tabel;*
- `<tip_data>` - *indică tipului câmpului;*
- `DEFAULT <expr>` - definirea valorii implicite pe care o ia expresia pentru coloană.

Constrângerile pot fi clasificate în:

- constrângeri de coloană, care definesc valorile pe care le poate lua o coloană sau mai multe coloane (*DEFAULT*, *NOT NULL*, *UNIQUE*, *CHECK*);
- constrângeri de integritate a tabelului, precizând cheia primară a acestuia (*PRIMARY KEY*);
- constrângeri de integritate referențială, care asigură coerența dintre cheile primare (sau unice) şi cheile externe corespunzătoare (*FOREIGN KEY*).

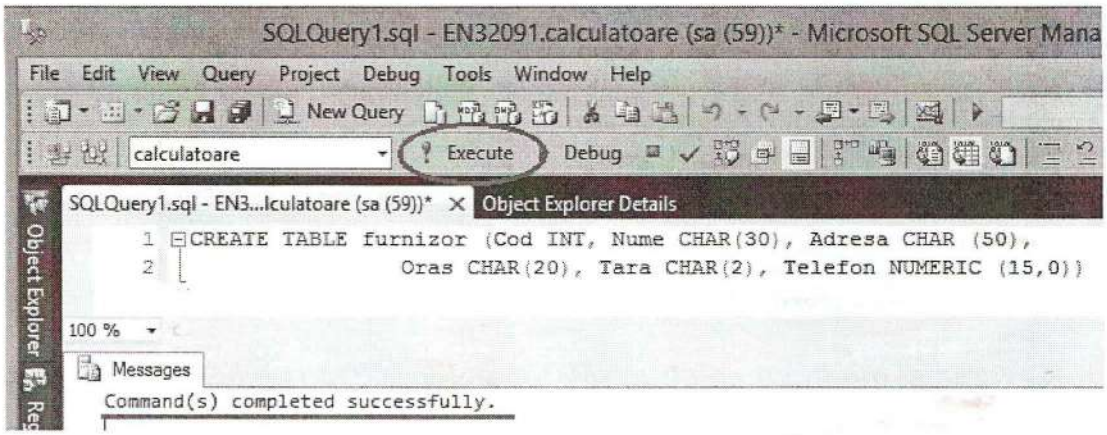
Pe de altă parte, constrângerile se pot clasifica după nivelul la care sunt definite în:

- constrângeri la nivel de coloană;
- constrângeri la nivel de tabel, care pot acționa asupra unei combinații de coloane.

Constrângerile impuse fiecărei coloane, ca şi constrângerile de tabel, sunt opționale. Constrângerile *NOT NULL* şi *DEFAULT* se pot defini doar la nivel de coloană. Constrângerile *UNIQUE*, *PRIMARY KEY*, *FOREIGN KEY* şi *CHECK* pot fi definite atât la nivel de coloană, cât şi la nivel de tabel. Totuşi, dacă aceste constrângeri implică mai multe coloane, atunci trebuie să fie definite obligatoriu la nivel de tabel.

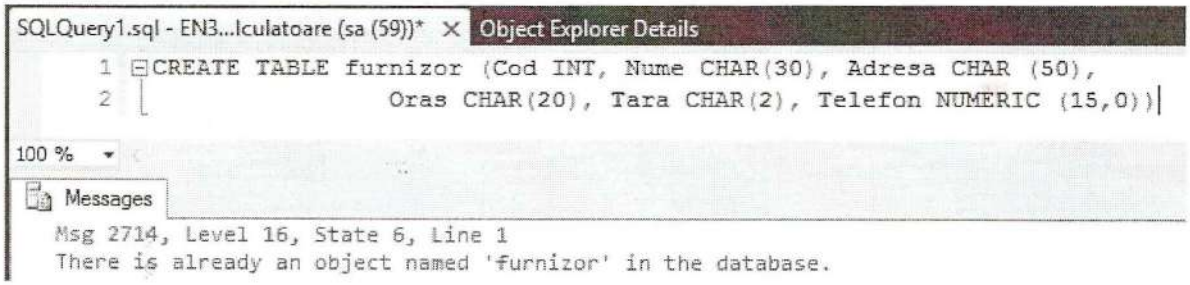
Aceasta este sintaxa de bază a instrucțiunii pentru crearea unei structuri de tabel în SQL. Există sintaxe mai complexe, la care se va reveni mai târziu.

În ***Transact-SQL Query Editor*** se inserează primul exemplu de creare a unui tabel. Fie în baza de date ***calculatoare*** trebuie creat un tabel numit ***furnizor*** cu câmpurile ***Cod***, ***Nume***, ***Adresa***, ***Oras***, ***Tara***, ***Telefon***. Să se observe cum este prezentată instrucțiunea în Query Editor:



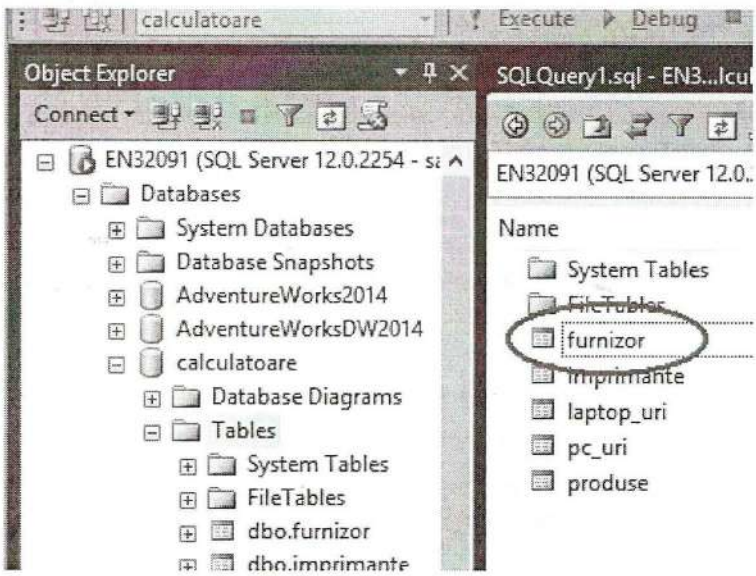
Nu înainte de scrierea instrucțiunii, se face un click pe comanda ***Execute*** sau tasta **F5**. După finalizarea procesării instrucțiunii, sistemul ne comunică faptul că instrucțiunea a fost încheiată cu succes şi tabelul a fost creat în baza de date respectivă.

În cazul unui nou click pe butonul de executare, va fi prezentat următorul mesaj de eroare:



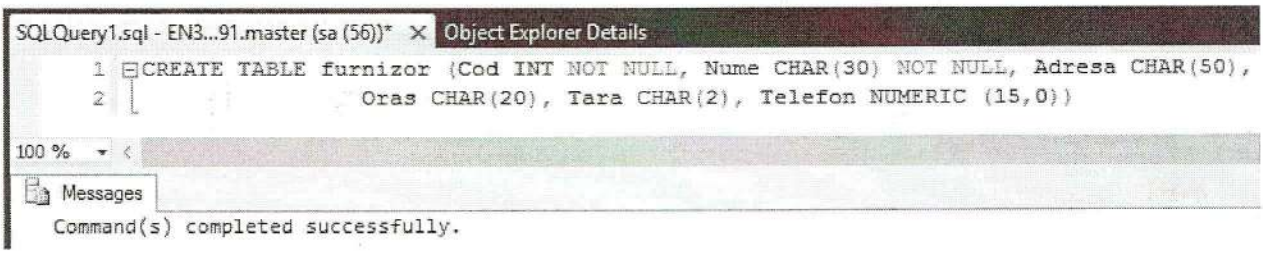
Mesajul comunică faptul că a avut loc o tentativă de creare a unui tabel cu un nume, deja, existent în baza de date în uz.

Utilizând ***Microsoft SQL Server Management Studio***, se poate verifica dacă tabelul a fost creat realmente. Pentru aceasta, se extind serverul, baza de date, unde a fost creat tabelul, precum şi dosarul ***Tables***:



Să se observe în ***Transact-SQL Query Editor*** că tabelul (încercuit) a fost creat pe server, în cadrul bazei de date specificate.

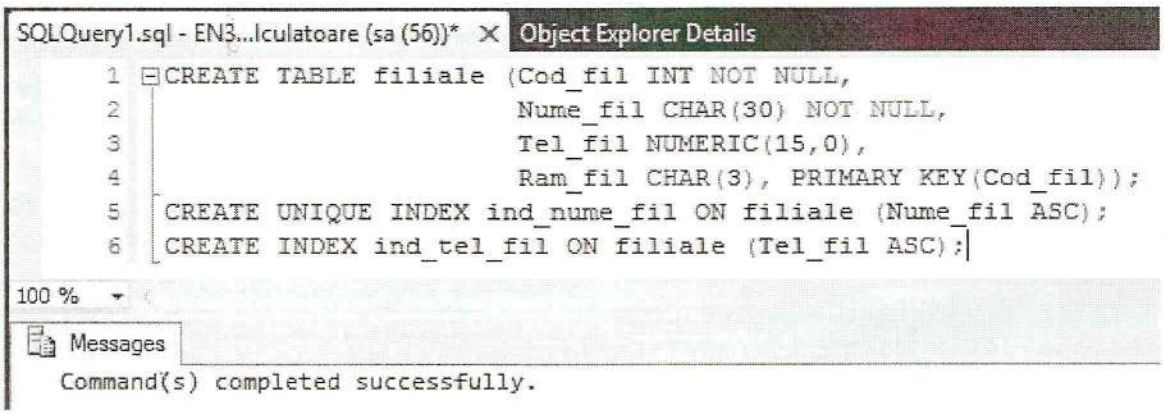
Acum se formulează următoarea instrucţiune:



Este evident că acest exemplu este similar celui anterior, cu o mică deosebire: pentru acelaşi tabel, s-a stabilit condiţia că câmpurile ***Cod*** şi ***Nume*** nu pot avea valori necunoscute (*NOT NULL*). Astfel, dacă se doreşte precizarea faptului că un câmp nu trebuie să conţină valori nule, este suficientă plasarea clauzei respective imediat după descrierea caracteristicilor câmpului, aşa cum e arătat în instrucţiunea de mai sus.

În continuare, se examinează o instrucţiune care creează structura unui tabel şi acesta posedă o cheie primară şi un index. Se analizează codul pas cu pas:

- *CREATE TABLE filiale(Cod_fil INT NOT NULL, Nume_fil CHAR(30) NOT NULL, Tel_fil NUMERIC(15,0), Ram_fil CHAR(3), PRIMARY KEY(Cod_fil))* - prima instrucţiune creează structura tabelului, unde câmpul ***Cod_fil*** este definit în calitate de cheie primară.
- *CREATE UNIQUE INDEX ind_nume_fil ON filiale(Nume_fil ASC)* – această linie creează un index pentru coloana ***Nume_fil*** şi o constrângere *(UNIQUE CONTRAINTS)*, care indică faptul că această coloană nu poate lua valori duplicate.
- *CREATE INDEX ind_tel_fil ON filiale(Tel_fil ASC)* - ultima linie creează un index pentru coloana ***Tel_fil***, numit ***ind_el_fil***.



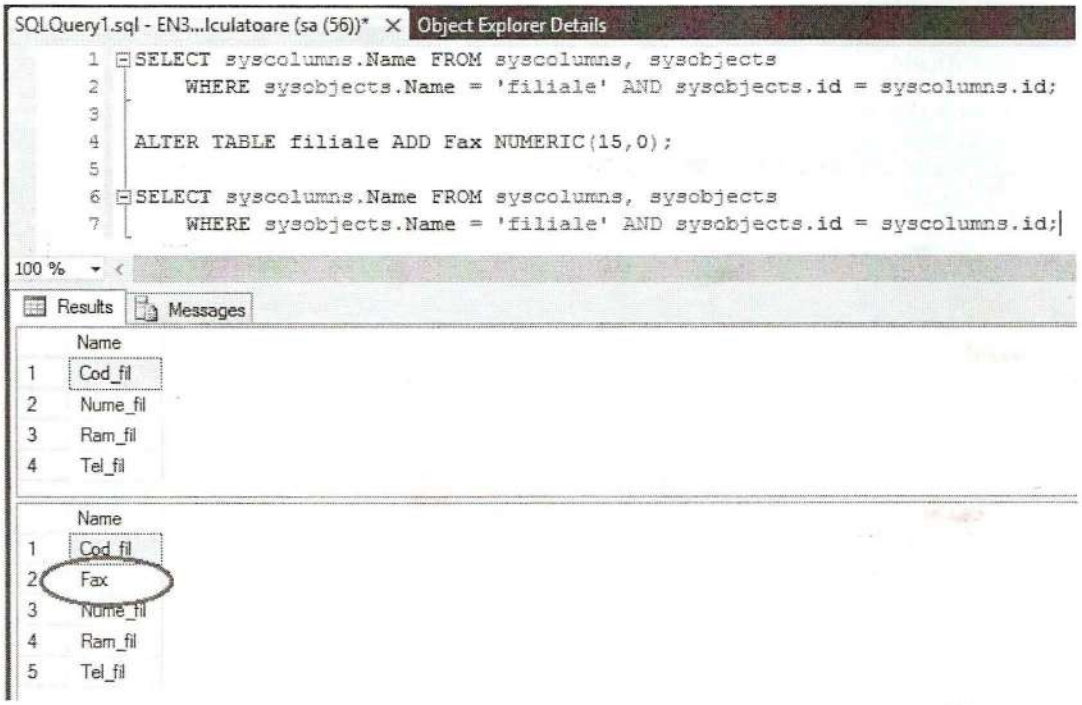
Astfel, a fost creat un tabel cu o cheie primară şi doi indecşi, din care unul defineşte o constrângere *UNIQUE* pe o coloană.

5.1.2. Instrucţiunile ALTER TABLE şi DROP TABLE

Uneori, apar situaţii când trebuie create noi tabele prin adăugarea coloanelor sau modificarea structurii tabelului. Modificarea structurii unui tabel se face folosind instrucţiuni *ALTER TABLE*. Forma instrucţiunii depinde de tipul modificării aduse:

```
ALTER TABLE <nume_tabel> <ACŢIUNE> <descriere>,
```

unde *<ACŢIUNE>* poate fi una din opţiunile *ADD*, *ALTER* sau *DROP*, iar *<descriere>* este o comandă de modificare asociată acţiunii. Modificarea unui tabel poate crea probleme, dacă este incompatibilă cu conţinutul acestuia. De exemplu, trecerea unui câmp la opţiunea *NOT NULL* impune faptul că această coloană are, deja, valori nenule pentru toate liniile tabelului.



Sintaxa pentru adăugarea unor noi coloane (nu se pot specifica poziţiile coloanelor noi în structura tabelului; coloanele noi devin automat ultimele în cadrul structurii tabelului):

```
ALTER TABLE <nume_tabel> ADD
  (<coloana> <tip_data> [DEFAULT <expr>] [, ...]);
```

De exemplu, instrucţiunea de adăugare a coloanei ***Fax,*** de tipul *NUMERIC,* în tabelul ***filiale,*** se scrie în felul următor:

```
ALTER TABLE filiale ADD Fax NUMERIC(15,0)
```

Instrucţiunea *SELECT,* executată înainte şi după *ALTER TABLE,* afişează schimbările ce au avut loc în tabelul ***filiale***. Tabelele de sistem ***sysobjects*** şi ***syscolumns*** conţin metadate despre obiectele şi coloanele bazei de date în uz.

Sintaxa pentru modificarea unor coloane (schimbarea tipurilor de date, a dimensiunilor sau a valorilor implicite ale acestora; schimbarea valorilor implicite afectează numai inserările care succed modificarea):

```
ALTER TABLE <nume_tabel> ALTER COLUMN
  (<coloana> <tip_data> [DEFAULT <expr>] [, ...]);
```

Următorul exemplu modifică caracteristicile coloanei ***Nume_fil*** din structura tabelului ***filiale:***

```
ALTER TABLE filiale ALTER COLUMN Nume_fil VARCHAR(50) NOT NULL
```

Pentru ştergerea unei coloane dintr-un tabel, în instrucţiunea *ALTER TABLE,* se foloseşte cuvântul-cheie *DROP*. Sintaxa instrucţiunii de eliminare a unei coloane din structura tabelului:

```
ALTER TABLE <nume_tabel> DROP COLUMN <coloana>;
```

De exemplu, ştergerea coloanei ***Tel_fil*** din tabelul ***filiale*** se poate face cu instrucţiunea:

```
ALTER TABLE filiale DROP COLUMN Tel_fil
```

Sunt necesare unele observaţii:

- Dimensiunea unei coloane numerice sau de tip caracter poate fi mărită, dar nu poate fi micşorată decât dacă acea coloană conţine numai valori *NULL* sau dacă tabelul nu conţine nicio linie.
- Tipul de date al unei coloane poate fi modificat doar dacă valorile coloanei respective sunt *NULL.*

- O coloană *CHAR* poate fi convertită la tipul de date *VARCHAR* sau invers, numai dacă valorile coloanei sunt *NULL* sau dacă nu se modifică dimensiunea coloanei.
- În afară de aceasta, înainte de a lansa o instrucțiune de modificare sau ştergere a unui tabel, trebuie să se țină cont de alte obiecte cu care acest tabel este legat şi de constrângerile definite în tabel.

În cazul în care se vrea ştergerea sau modificarea unui tabel trebuie să fim siguri de ceea ce se face, deoarece nu se poate recupera ce s-a şters sau ce s-a modificat.

Un tabel poate fi desființat, dacă nu este nevoie de el un timp mai îndelungat. Ştergerea fizică a cestuia, inclusiv a înregistrărilor acestuia, se realizează prin comanda:

```
DROP TABLE <nume_tabel>
```

Pentru ştergerea conținutului unui tabel şi păstrarea structurii acestuia se poate utiliza instrucțiunea:

```
TRUNCATE TABLE <nume_tabel>
```

Următorul exemplu va şterge tot conținutul tabelului ***filiale***:

```
TRUNCATE TABLE filiale;
```

5.2. Instrucțiunile de actualizare a bazei de date

Limbajul Transact-SQL propune trei instrucțiuni ce corespund celor trei operații de actualizare a bazei de date: inserarea (*NSERT*), suprimarea (*DELETE*) şi modificarea (*UPDATE*). Toate aceste trei operații se aplică asupra tabelelor (fizice sau virtuale) şi actualizarea se petrece la nivel de linie de tabel.

5.2.1. Instrucțiunea INSERT

Este evident că, dacă pot fi create tabele prin cod, pot fi şi inserate date prin cod. Datele pot fi inserate într-o linie întreagă sau o parte a acesteia, iar restul câmpurile trebuie să posedă proprietatea *NOT NULL* pentru a fi completate şi evitate erorile logice.

Pentru inserarea datelor, se utilizează instrucțiunea *INSERT*. Sintaxa de bază a acestei instrucțiuni este:

```
INSERT [INTO] <nume tabel> [(<coloana1> [, <coloana2> ... ])]
       {VALUES (<expresie1> [,<expresie2> ... ])
        |<SELECT ... FROM ...>
       };
```

Argumentele folosite sunt:

- `<nume tabel>` - numele tabelului la care sunt adăugate înregistrările;
- `[(<coloana1> [, <coloana2> ... ])` - lista coloanelor pentru care se introduc valori;
- `(<expresie1> [,<expresie2> ... ])` - sunt valorile asociate expresiilor în ordinea coloanelor;
- `<SELECT ... FROM ...>` - interogare *SELECT*.

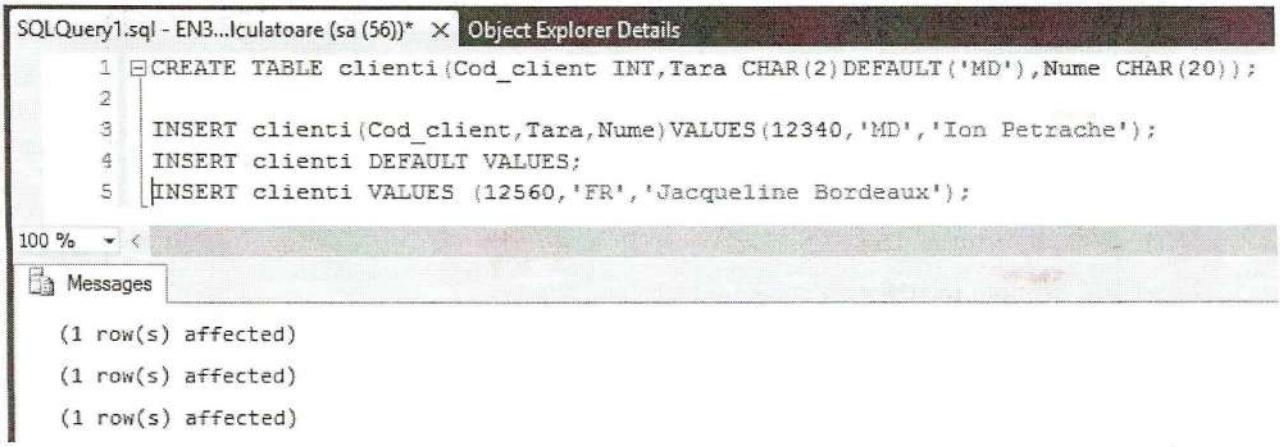
Între valori şi numele de coloane, trebuie să existe o corespondență biunivocă. Lista de coloane poate sa lipsească, dacă se introduc valori în toate coloanele tabelului, dar, în această situație ordinea valorilor introduse trebuie să respecte ordinea coloanelor.

Coloanele pot fi precizate în orice ordine, însă trebuie asigurată corespondența între numele coloanelor şi valorile furnizate. În afară de aceasta, dimensiunile datelor introduse trebuie să fie mai mici decât dimensiunea maximă a coloanei în care se inserează. Prin forma *INSERT ...VALUES*, se introduce în tabel o singură înregistrare. Cu ajutorul valorii *NULL*, se pot introduce valori nule.

Există şi a doua formă de inserare a datelor în tabele.

Se scrie instrucțiunea *INSERT* cu ajutorul unei interogări *SELECT*. În acest caz, nu se utilizează clauza *VALUES*. Totodată, între clauza *INSERT* şi interogarea *SELECT*, trebuie să existe compatibilitate de tip, număr şi ordine între lista de valori şi lista de câmpuri corespondente din tabel.

Se examinează exemplul următor:



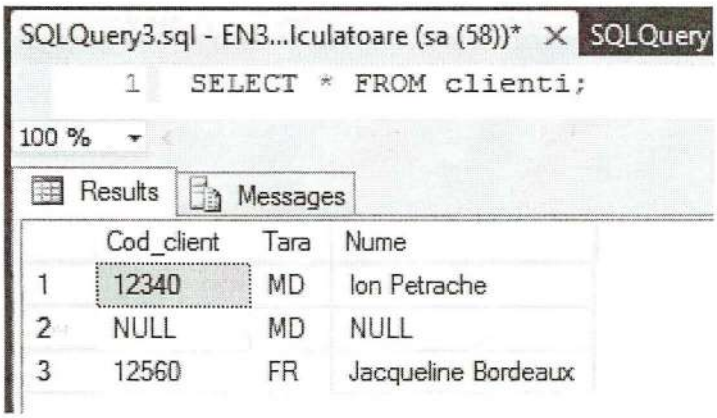
Prima instrucțiune creează tabelul ***clienti*** cu coloanele ***Cod_client***, ***Tara*** şi ***Nume***. Abordarea constă în faptul că pentru câmpul ***Tara*** este definită valoarea predeterminată MD.

Instrucțiunea a doua inserează o înregistrare (linie) care are câmpul ***Cod_client***=*12345*, câmpul ***Tara***=*'MD'* şi câmpul ***Nume***=*'Ion Petrache'*.

Instrucțiunea a treia inserează o înregistrare numai cu valorile implicite, definite în structura tabelului, adică câmpurile ***Cod_client*** şi ***Nume*** vor conține valoarea *NULL*, iar câmpul ***Tara***=*'MD'*, deoarece *'MD'* este o mărime predefinită.

Şi ultima instrucțiune inserează o înregistrare cu ***Cod_client***=*12351*, câmpul ***Tara***=*'FR'* şi câmpul ***Nume***=*'Jacqueline Bordeaux'*. Ce se întâmplă, dacă nu sunt specificate denumirile câmpurilor? În acest caz, SQL inserează valorile în câmpuri în ordinea în care acestea au fost descrise în structura tabelului.

Să se examineze acum următoarea interogare asupra tabelului în cauză:



Interogarea construită solicită să fie afişate toate coloanele tabelului ***clienti***. După acționarea butonului **Execute,** se obține rezultatul din fereastra de mai sus.

Se afişează încă un exemplu, acum cu un argument nou în instrucțiunea *CREATE*:

```
SQLQuery3.sql - EN3...lculatoare (sa (57))*  |  Object Explorer Details

1  DROP TABLE clienti;
2
3  CREATE TABLE clienti(Cod_client INT,Nume CHAR(20),Tara CHAR(2)
4      CHECK (Tara IN ('MD','RO','FR','GB'))DEFAULT ('MD'));
5
6  INSERT clienti VALUES(12200,'Ion Petrache','MD');
7  INSERT clienti VALUES(12202,'Jacqueline Bordeaux','FR');
8  INSERT clienti VALUES (12204,'Sergey Mihailov','RU');
9
10 SELECT * FROM clienti;

100 %

Results | Messages

(1 row(s) affected)

(1 row(s) affected)
Msg 547, Level 16, State 0, Line 8
The INSERT statement conflicted with the CHECK constraint "CK__clienti__Tara__6383C8BA".
The conflict occurred in database "calculatoare", table "dbo.clienti", column 'Tara'.
The statement has been terminated.

(2 row(s) affected)
```

Urmează descrierea tuturor instrucţiunile din exemplul de mai sus:

Chiar de la început se observă o schimbare, instrucţiunea *DROP TABLE*, care are funcţia de a suprima structura tabelului specificat. Nu poate fi creată o structură nouă a tabelului ***clienti***, fără a fi exclusă cea veche din schema bazei de date.

Mai departe, se creează un tabel ***clienti*** cu câmpurile ***Cod_client***, ***Nume*** şi ***Tara***. În structura acestui tabel, este introdusă o altă noutate: pentru câmpul ***Tara*** este inserată clauza *CHECK* care acceptă pentru acest câmp valori constante dintr-o listă dată, adică '*MD*', '*RO*', '*FR*', '*GB*'. Valoarea predefinită pentru acest câmp este '*MD*'.

În continuare, sunt formulate o serie de instrucţiuni *INSERT*. Instrucţiunea a treia *INSERT* face tentativa de a insera o înregistrare nouă cu valoarea '*RU*' pentru atributul ***Tara***, care nu este definită în clauza *CHECK*. Această linie va fi respinsă în mod automat de SQL şi, precum se poate observa, sistemul a generat un mesaj de eroare, prezentat în fereastra de rezultate. Prin urmare, au fost confirmate de SQL numai primele două instrucţiuni *INSERT*.

În final, este scrisă instrucţiunea *SELECT*, care afişează toate înregistrările tabelului construit.

Se construieşte un alt exemplu, folosind instrucţiunile *CREATE* şi *INSERT*.

```
SQLQuery3.sql - EN3...lculatoare (sa (57))*  |  Object Explorer Details

1  CREATE TABLE Vanzari(Numar_de_rand INT NOT NULL IDENTITY(1,1),
2                 Cod_client INT NOT NULL,
3                 Data_vanzare DATETIME NOT NULL
4                    CONSTRAINT DATETIME DEFAULT (GETDATE()));
5
6  INSERT vanzari (Cod_client)VALUES(001);
7  INSERT vanzari (Cod_client)VALUES(045);
8
9  SELECT * FROM vanzari;

100 %

Results | Messages
```

	Numar_de_rand	Cod_client	Data_vanzare
1	1	1	2015-02-05 01:59:23.070
2	2	45	2015-02-05 01:59:23.073

Exemplul introduce mai multe elemente în ceea ce priveşte instrucţiunea *CREATE TABLE*. Astfel, se solicită crearea unui tabel numit ***vanzari*** cu câmpurile:

- ***Numar_de_rand***, care nu poate fi nul şi este generat automat de clauza *IDENTITY*, pornind de la numărul 1 şi având o creştere de +1 la fiecare înregistrare nouă.
- ***Cod_client***, fără mari schimbări.
- ***Data_vanzare***, care nu poate poseda valori necunoscute, este generată automat prin atribuirea rezultatului funcţiei *GETDATE()*, care returnează data şi ora sistemului.

Comenzile *INSERT* sunt urmate de o instrucţiune de afişare a tabelului.

Dacă într-un tabel se introduc linii în *INSERT* prin intermediul unei subinterogări, coloanele din lista *SELECT* trebuie să corespundă, ca număr şi tip, celor precizate în clauza *INTO*. În absenţa unei liste de coloane în clauza *INTO*, subinterogarea trebuie să furnizeze valori pentru fiecare coloană a obiectului destinaţie, respectând ordinea în care acestea au fost definite. Astfel, este posibil să se copieze mai multe înregistrări dintr-un tabel într-unul sau mai multe tabele.

Se presupune că există un tabel ***clienti_MD***, care are aceeaşi structură ca şi tabelul ***clienti***. Să se insereze, în acest tabel, datele clienţilor din Republica Moldova. Instrucţiunea va avea forma:

```
INSERT INTO clienti_MD
    (SELECT * FROM clienti WHERE Tara='MD');
```

În cazul în care coloanele celor două tabele au aceeaşi specificare, definirea listei de coloane în instrucţiunea *INSERT* este opţională.

5.2.2. Instrucţiunea UPDATE

Instrucţiunea *UPDATE* permite actualizarea valorilor coloanelor dintr-una sau mai multe linii ale unui tabel. Modificarea datelor dintr-un tabel utilizează una din formele sintactice ale comenzii:

```
UPDATE <nume tabel>
    SET <coloana1> = <expresie1> [,<coloana2> = <expresie2>, ... ]
    [WHERE <criteriu_de_modificare>];
```

Formatul fundamental al unei instrucţiuni *UPDATE* este alcătuit din patru componente:

- `<nume tabel>` - reprezintă denumirea tabelului în care se va realiza instrucţiunea de modificare;
- `(<coloana1> [, <coloana2> ... ])` - sunt coloane ale tabelului specificat;
- `(<expresie1> [,<expresie2> ... ])` - sunt valorile corespunzătoare coloanelor;
- `<criteriu_de_modificare>` - condiţia de filtrare care determină liniile ce urmează a fi actualizate.

Există două posibilităţi de modificare. Una constă în furnizarea, în mod explicit, a fiecărei valori pentru coloanele care trebuie modificate, iar cealaltă posibilitate constă în obţinerea valorilor în urma unei interogări *SELECT*.

Folosind prima posibilitate, clauza *SET* atribuie coloanei valoarea specificată. Când se modifică coloane multiple, se foloseşte o singură comandă *SET*, iar fiecare pereche `<coloana> = <expresie>` este separată de următoarea printr-o virgulă.

Clauza *WHERE* impune ca actualizarea valorilor coloanelor să se efectueze numai asupra acelor linii care îndeplinesc condiţia dată. Dacă este omisă clauza *WHERE*, atunci vor fi modificate valorile coloanelor din toate liniile tabelului.

Este examinată secvenţa de instrucţiuni de mai jos, unde, la început, este creată structura tabelului în care sunt inserate cinci înregistrări:

```
SQLQuery3.sql - EN3...lculatoare (sa (57))*  X  Object Explorer Details
    1  CREATE TABLE functionari(Cod INT NOT NULL,Nume CHAR(20),Sector CHAR(13));
    2
    3  INSERT functionari VALUES(001,'Paul Mateescu','Suport tehnic');
    4  INSERT functionari VALUES(002,'Gheorghe Goldea','Administrare');
    5  INSERT functionari VALUES(003,'Isabela Arama','Finante');
    6  INSERT functionari VALUES(004,'Radu Arama','Finante');
    7  INSERT functionari VALUES(005,'Ana Mateescu','Administrare');
    8
    9  SELECT * FROM functionari;
100 %
Results
(1 row(s) affected)
(1 row(s) affected)
(1 row(s) affected)
(1 row(s) affected)

Cod         Nume                 Sector
----------- -------------------- -------------
1           Paul Mateescu        Suport tehnic
2           Gheorghe Goldea      Administrare
3           Isabela Arama        Finante
4           Radu Arama           Finante
5           Ana Mateescu         Administrare

(5 row(s) affected)
```

Rezultatul comenzii *SELECT* este prezentat sub formă de text. Acest model de vizionare poate fi ales, dacă se apasă butonul ***Results to text.***

Nimic deosebit nu se observă în cele scrise mai sus. Acum, se presupune că toţi funcţionarii sectorului financiar au fost transferaţi în sectorul de administrare. Atunci instrucţiunea va avea următoarea structură şi următorul rezultat:

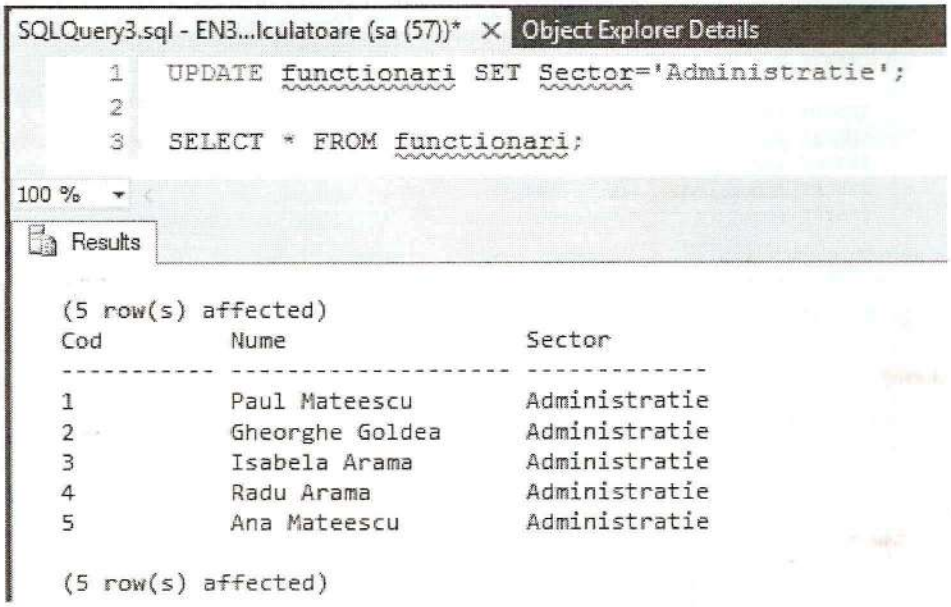
```
SQLQuery3.sql - EN3...lculatoare (sa (57))*  X  Object Explorer Details
    1  UPDATE functionari SET Sector='Administratie' WHERE Sector='Finante';
    2
    3  SELECT * FROM functionari ORDER BY Sector;
100 %
Results
(2 row(s) affected)
Cod         Nume                 Sector
----------- -------------------- -------------
2           Gheorghe Goldea      Administrare
5           Ana Mateescu         Administrare
3           Isabela Arama        Administratie
4           Radu Arama           Administratie
1           Paul Mateescu        Suport tehnic

(5 row(s) affected)
```

În structura acestei instrucţiuni, pentru prima dată, este folosit cuvântul rezervat *UPDATE,* care pregăteşte tabelul pentru suportarea unor modificări.

În continuare, se observă că este modificat câmpul ***Sector*** cu conţinutul *'Administrare'* pentru toate înregistrările în care câmpul ***Sector*** este egal cu *'Finante'*. Şi, în sfârşit, este solicitată o afişare a înregistrărilor tabelului de funcţionari, ordonat pe câmpul ***Sector.***

Precum se poate vedea, modificarea datelor unui tabel este destul de simplă, dacă se face abstracţie de condiţiile impuse de clauza *WHERE.* Astfel, rezultatul aceleiaşi instrucţiuni, fără clauza *WHERE,* este reprezentat în următoarea fereastră:



În structura instrucțiunii, lipseşte clauza *WHERE*, care acționează în calitate de filtru. În acest caz, automat, toate înregistrările în coloana ***Sector*** au luat valoarea egală cu '*Administrare*'.

5.2.3. Instrucțiunea DELETE

În bazele de date, din operațiile de actualizare, în afară de inserare şi modificare a liniilor, face parte şi operația de eliminare a conținutului uneia sau mai multor linii.

Instrucțiunea *DELETE* permite ştergerea liniilor dintr-un tabel şi are următoarea sintaxă:

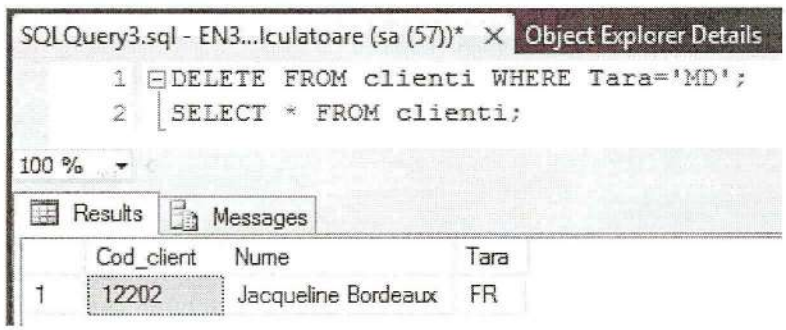
```
DELETE [FROM] <nume tabel>
       [WHERE <criteriu_de_ştergere>];
```

unde

- `<nume tabel>` - reprezintă denumirea tabelului în care se va realiza operația de ştergere;
- `<criteriu_de_ştergere>` - este o expresie logică, în care intervin valorile unor coloane ale tabelului.

Folosirea clauzei *WHERE* determină ştergerea acelor linii care îndeplinesc condiția impusă. În această clauză, pot fi folosite şi subcereri. Dacă nu este specificată nicio condiție, se şterg toate liniile tabelului.

Astfel, instrucțiunea *DELETE* este destul de simplă:



Instrucțiunea *DELETE* determină ştergerea, din tabelul ***clienti***, tuturor înregistrările în care câmpul ***Tara*** este egal cu '*MD*'. Se poate reveni la pagina precedentă, pentru a verifica că, în lista nouă, nu există înregistrarea care are câmpul ***Nume***='*Ion Petrache*', deoarece, pentru această înregistrare, câmpul ***Tara***='*MD*'.

Dacă este omisă clauza *WHERE*, atunci vor fi şterse toate liniile din tabel. Iată câteva puncte de care trebuie să se ţină cont în utilizarea instrucţiunii *DELETE*:

- Instrucţiunea *DELETE* este mai sigură tranzacţional, dar este mai lentă decât instrucţiunea *TRUNCATE TABLE*, deoarece *TRUNCATE TABLE* şterge un tabel şi îl recreează.
- În cazul instrucţiunii *DELETE*, numărul liniilor şterse poate fi returnat, spre deosebire de utilizarea instrucţiunii *TRUNCATE TABLE*.
- Declanşatoarele *ON DELETE* nu sunt invocate, deoarece *TRUNCATE* nu utilizează instrucţiunea *DELETE*.

5.3. Gestiunea indecşilor în Microsoft SQL Server Management Studio

Acum, se va trece la sarcina creării indecşilor, un mecanism informatic utilizat în bazele de date [4].

Mai întâi, e foarte important să se cunoască ce este un index propriu-zis. Un index este o structură pe disc asociată tabelului sau viziunii pentru căutarea rândurilor în tabel sau viziune. Indexul conţine chei construite pe una sau mai multe coloane din tabel sau viziune. Aceste chei sunt păstrate într-o structură (B-arbore) care permite sistemului SQL Server să găsească în mod rapid şi eficient rândul sau rândurile asociate valorii pe care a luat-o cheia.

5.3.1. Tipurile de indecşi utilizaţi în SQL Server 2014

Astfel, indexul este o structură legată de un tabel care are drept scop reducerea timpului de acces la înregistrările tabelului. În SQL Server, se utilizează câteva tipuri de indecşi:

`Clustered`

Acest tip de index organizează liniile tabelelor în aceeaşi ordine, ca şi intrările în index. Este tipul de index ideal, care nu implică multe actualizări în tabel, deoarece, pentru fiecare modificare a conţinutului unui câmp-cheie, tabelul este reorganizat. Fiecare tabel poate conţine numai un index de acest tip. Există un singur caz când liniile într-un tabel sunt sortate într-o ordine - atunci când tabelul conţine un index de tip *Clustered*. Dacă tabelul conţine un index *Clustered*, tabelul este numit – tabel clasterizat. Dacă tabelul nu posedă acest index, atunci liniile acestui tabel reprezintă o structură neordonată, care se numeşte *heap*.

`Nonclustered`

Acest tip de index are o structură separată de cea a tabelului. Ordinea fizică a liniilor tabelului nu corespunde ordinii înregistrărilor fişierului index. Poate fi comparat cu indexul unei cărţi, unde indexul este la sfârşitul ediţiei, iar datele - la începutul ei. Un tabel poate avea mai mulţi indecşi de acest tip. Pointerul dintr-o linie-index a indexului *Nonclustered* către o linie de date se numeşte locator de linie. Structura locatorului de linie depinde de modul de aranjare a paginilor lor de date, sau ele sunt stocate într-un tabel-*heap* sau într-un tabel clasterizat. Pentru un *heap*, un locator de linie este un pointer spre linie. Pentru un tabel clasterizat, un locator de linie este cheia indexului clasterizat

`Covering Indexes`

Index de acoperire – este un index *Nonclustered* care conţine toate datele necesare pentru a satisface o interogare. Indexul de acoperire reprezintă o strategie pentru îmbunătăţirea performanţelor de interogare şi este extins cu coloane ce nu fac parte din cheie suplimentar la coloanele ce constituie cheia. Conceptul de coloane incluse a fost introdus în SQL Server 2005. Acestea permit administratorului bazei de date să stocheze date despre coloanele tabelului în paginile indexului *Nonclustered* şi evită căutările costisitoare a înregistrării necesare în indexul *Clustered*. Desigur că în orice optimizare a indexului, trebuie să se ţină cont că coloanele noi incluse (ce nu fac parte din cheie) necesită o mentenanţă suplimentară în cazul în care datele din tabel sunt modificate.

Unique

Indexul de acest tip exclude cazul când cheia indexului are valori duplicate şi, prin urmare, fiecare rând în tabel sau în viziune este, de asemenea, unic. Ambele tipuri de indecşi *Clustered* şi *Nonclustered* pot fi *Unique*. Aceasta presupune că oricare două rânduri nu pot avea aceleaşi valori pe câmpurile ce constituie cheia indexului. În caz contrar, indexul nu este de tip *Unique* şi mai multe rânduri pot avea aceeaşi valoare pentru cheie.

Indexed views

Un index asupra viziunii este materializat (executat). Adică viziunea şi mulţimea-rezultat este permanent păstrată într-un index de tipul *Unique Clustered,* în acelaşi mod, precum sunt păstrate tabelul şi indexul *Clustered.* Indecşii de tip *Nonclustered* asupra viziunilor pot fi adăugaţi după crearea indecşilor de tip *Clustered.*

Spatial

Un index spaţial oferă posibilitatea de efectuare mai eficientă a unor operaţii asupra obiectelor spaţiale (datelor spaţiale) dintr-o coloană de tip *geometry* sau *geography*. Un index spaţial reduce numărul de obiecte asupra cărora trebuie să fie aplicate operaţiile spaţiale care sunt relativ mai costisitoare (consumatoare de surse hardware).

Filtered

Reprezintă un index nonclustered optimizat, special adaptat pentru a satisface interogările care selectează date dintr-o submulţime de date bine definită. Acesta, pentru a indexa o porţiune de linii din tabel, utilizează un filtru sub forma unui predicat. Un index de tip *filtered* bine conceput poate îmbunătăţi performanţele interogării, reducând costurile de întreţinere a indexului şi costurile de depozitare, comparativ cu indexul *full-tabel.*

Full-text

Un tip special de index funcţional orientat pe marcare, care este construit şi menţinut de ***Microsoft Full-Text Engine for SQL Server (MSFTESQL) service.*** El oferă un suport eficient pentru căutarea sofisticată a cuvintelor în datele de tip secvenţe de caractere.

Primary XML and Secondary XML Indexes

Tipul de date XML a fost introdus în SQL Server 2005. El permite stocarea documentelor XML sau fragmentelor XML într-o coloană de tabel. Pentru oferirea unei performanţe rezonabile la interogarea datelor XML, se creează unul sau mai mulţi indecşi XML. Dacă indecşii XML nu sunt creaţi, întregul document XML trebuie să fie încărcat în ***XML Document Object Model*** (DOM) şi căutat. O aplicare corectă a indecşilor XML, evită necesitatea creării XML DOM pentru fiecare rând care se examinează.

Columnstore Indexes

Reprezintă un nou tip de indecşi, apărut pentru prima dată în SQL Server 2012 ca indecşi non-clustered. În versiunea 2014, indecşii *columnstore* pot fi şi de tip clustered. Acest tip de indecşi nu utilizează B-arborii. Indecşii de acest tip au fost introduşi pentru a oferi performanţe înalte interogărilor care, de obicei, sunt tipice magaziilor de date construite pe agregări ale seturilor mari de date.

Fiecare tabel poate conţine maximum 249 de indecşi de tip *Nonclustered* şi un index de tipul *Clustered.* Dintre aceştia fac parte şi indecşii generaţi pentru a susţine constrângerile *PRIMARY KEY* şi *UNIQUE.* Un index de tip *Clustered* poate include maximum 16 coloane.

Una din principalele caracteristici ale SGBD-ului SQL Server constă în posibilitatea de creare în structura indexului a paginilor cu spaţiu pentru inserarea cheilor noi, fără a fi reorganizat, la nivel fizic, tot B-arborele. Atunci când un index de tipul *Clustered* este creat, SQL Server păstrează datele tabelului în paginile bazei de date în acord cu ordinea câmpurilor-cheie.

Dacă, în tabel, este inserată o linie nouă sau dacă valorile câmpului-cheie sunt modificate, este necesară reorganizarea liniilor tabelului în paginile bazei de date. Dacă o pagină este complet umplută, serverul mută aproximativ jumătate din liniile acestei pagini pe o pagină nouă.

Acest proces poartă numele de *page split* şi, în funcţie de dimensiunea tabelului, poate prejudicia sensibil executarea, în principal, dacă apar multe actualizări ale câmpurilor-cheie.

Indecşii care includ coloane ce nu fac parte din chei oferă un avantaj mare, dacă ei acoperă interogarea. Adică, dacă indecşii cuprind toate coloanele la care se referă interogarea.

Adăugarea coloanelor ce nu fac parte din cheie necesită spaţiu de disc suplimentar pentru păstrarea indexului. În particular, adăugarea coloanelor ce nu fac parte din cheie de tipul *VARCHAR(MAX), NVARCHAR(MAX), VARBINARY(MAX)* sau *XML* poate, esenţial, să crească necesităţile de spaţiu, deoarece valorile coloanelor sunt copiate în index şi rămân, de asemenea, în tabel sau în indexul clusterizat.

Beneficiul este atins în cazul operaţiilor de selectare, deoarece optimizatorul de interogări poate localiza datele din coloanele necesare în limitele indexului, nefiind accesate tabelul sau indexul de tip *Clustered*. Totuşi, existenţa multor coloane incluse poate mări timpul necesar pentru executarea operaţiilor de inserare, modificare şi ştergere a tabelului de bază sau a indexului viziunii, din cauza creşterii operaţiilor de menţinere a indexului.

Există trei metode de creare a indecşilor.

În primul rând, prin definirea unei constrângeri *PRIMARY KEY* sau *UNIQUE* asupra unei coloane, apelând la instrucţiunile *CREATE TABLE* sau *ALTER TABLE*. Sistemul SQL Server creează automat un index unic pentru a impune cerinţele de unicitate dictate de constrângerile *PRIMARY KEY* sau *UNIQUE*. În mod implicit, un index *unique clustered* este creat pentru a impune o constrângere *PRIMARY KEY*, cu excepţia cazului în care există deja un index *clustered*, sau se specifică un index *unique nonclustered*. În mod implicit, un index *unique nonclustered* este creat pentru a impune o constrângere *UNIQUE*, cu excepţia cazului în care un index *clustered* este specificat în mod explicit şi un index *clustered* nu există.

A doua metodă de creare a indecşilor se realizează prin intermediul instrucţiunilor Transact-SQL.

A treia metodă - folosind instrumentele grafice din ***SQL Server Management Studio***.

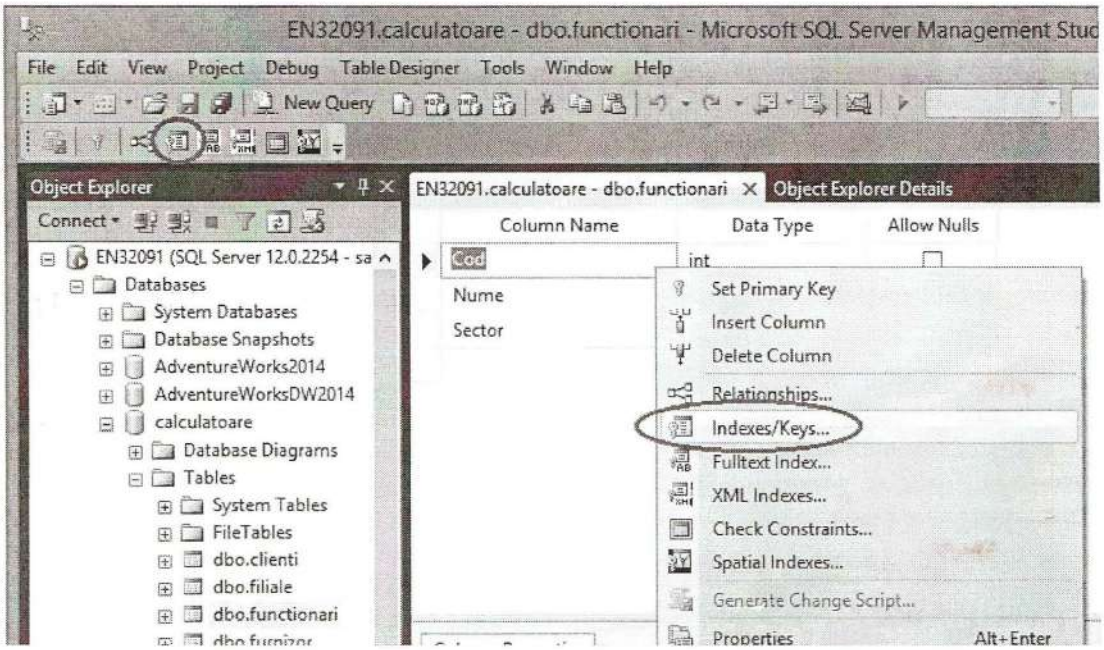
De la bun început, se trece la examinarea formei a treia de creare a indecşilor.

5.3.2. Crearea indecşilor asupra tabelelor existente

Pentru vizualizarea structurii unui tabel şi creării unui index, se execută următorii paşi:

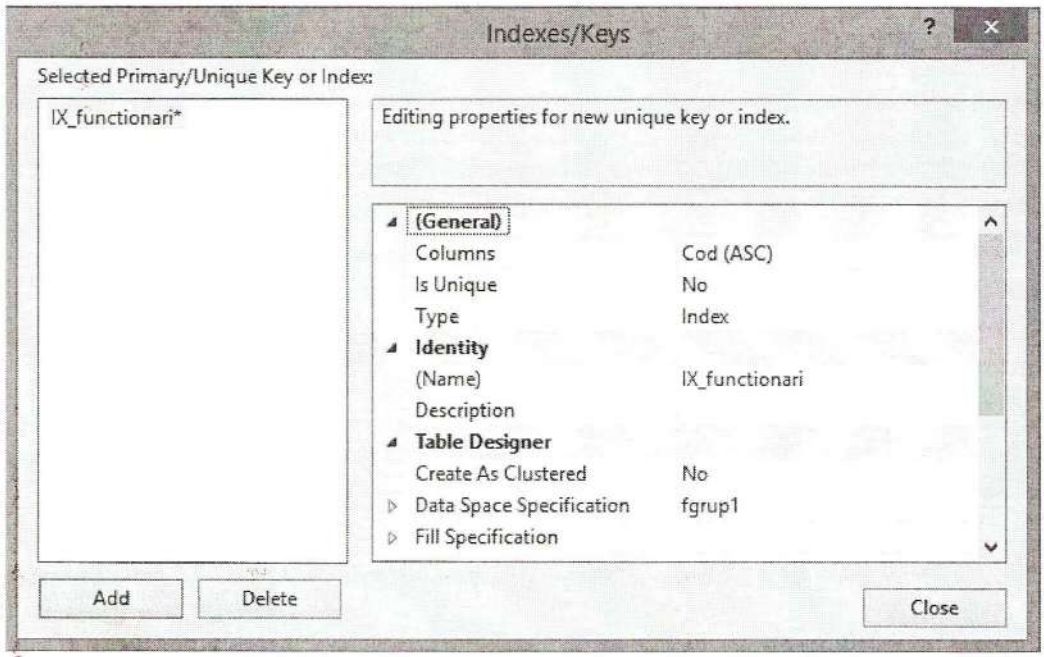
1. Se deschide ***Microsoft SQL Server Management Studio***.
2. În arborele ***Object Explorer,*** se extinde dosarul ***Databases***.
3. Se extinde dosarul bazei de date, unde va fi creat indexul.
4. Se extinde dosarul ***Tables***.
5. Se face un click-drept pe tabelul ales (fie ***functionari***). În meniul de context, se face un click pe opţiunea ***Design***.
6. Apare interfaţa deja cunoscută de proiectare în regim de constructor al tabelului.
7. Se face un click cu butonul drept al mouse-ului asupra oricărei proprietăţi a coloanelor.
8. În meniul de context apărut, se selectează opţiunea ***Indexes/Keys....***

   Pentru a obţine un rezultat similar, se pot omite paşii 7-8, dacă se apasă butonul ***Manage Indexes and Keys*** de pe bara de instrumente, aşa cum este prezentat în figura de mai jos:



9. În rezultat, apare o fereastră de dialog ***Indexes/Keys,*** separată vertical în două părți. Prima din stânga - ***Selected Primary/Unique Key or Index***: afişează cheile primare sau unice, sau indecşii. Se selectează una din ele pentru a prezenta proprietățile acesteia în gridul din dreapta. Dacă lista este vidă, nici un obiect din acestea nu a fost definit pentru tabelul dat. În cazul nostru, pentru a crea o entitate nouă, se apasă butonul **Add**.

   După apăsarea butonului **Add,** în fereastra din stânga, apar mai multe opțiuni pentru setarea proprietăților entității create. Pentru a şterge cheia sau indexul selectat în lista ***Selected Primary/Unique Key or Index,*** se apasă butonul **Delete**.



Proprietățile entității create sunt grupate pe categorii:

General

Dacă această categorie este expandată, sunt prezentate proprietățile ***Columns***, ***Is Unique*** şi ***Type***.

- ***Columns:*** prezintă ordinea coloanelor în cheie sau index şi asigură accesul la o casetă de dialog unde această ordine poate fi definită. Pentru a afişa caseta de dialog, se aplică un click pe ***Columns***, apoi pe butonul (...), care apare în partea dreaptă a câmpului acestei proprietăţi. În fereastra apărută ***(Index Column),*** se face un click în câmpul ***Colum Name***. Din meniul derulant apărut, se selectează câmpul (câmpurile) care va (vor) face parte din index. În cazul nostru, se selectează câmpul ***Cod***. Apoi se apasă butonul **OK** pentru confirmare.
- ***Is Unique:*** dacă se doreşte crearea unui index de tip *UNIQUE*, se marchează caseta de selecţie ***Is Unique*** şi apoi se alege una din opţiunile disponibile. Aceasta este indisponibilă pentru indecşii de tip XML.
- ***Type:*** specifică dacă entitatea selectată în lista ***Selected Primary/Unique Key or Index*** este o cheie *UNIQUE*, *PRIMARY* sau un index. Pentru cheia primară, acest câmp este numai de citire.

Identity

*Dacă această categorie este expandată, sunt afişate proprietăţile **Name** şi **Description**.*

- ***Name***: prezintă numele cheii sau al indexului. Atunci când este creată o entitate nouă, se atribuie un nume implicit bazat pe tabelul în fereastra activă în ***Table Designer.*** Numele *poate fi modificat în orice moment.*
- ***Description:*** oferă un loc pentru descrierea cheii sau indexului. Pentru a face o descriere mai detaliată, se aplică un click pe ***Description***, apoi un click pe butonul (...), care apare în partea dreaptă a câmpului al acestei proprietăţi. Aceasta oferă o arie mai largă în care poate fi inserat textul.

Table Designer

Dacă această categorie este expandată, este prezentată informaţia despre ***Create as Clustered***.

- ***Create as Clustered:*** face cheia sau indexul să fie clusterizat. Într-un tabel, este permis numai un index de tip *Clustered* Datele într-un tabel sunt păstrate în ordinea indexului de tip *Clustered.*
- ***Data Space Specification:*** dacă această opţiune este expandată, sunt prezentate informaţiile despre (Data Space Type), Filegroup or Partition Scheme Name şi Partition Column List.
  - ***(Data Space Type):*** indică dacă acest index sau cheie se referă la grupul de fişiere sau schema de partiţie.
  - ***Filegroup or Partition Scheme Name:*** prezintă numele grupului de fişiere sau al schemei de partiţie pe care este păstrat.
  - ***Partition Column List:*** afişează lista de coloane separate prin virgulă, care participă la funcţia de partiţie a coloanelor. Indisponibilă, dacă în câmpul ***(Data Space Type)*** este selectată opţiunea ***Filegroup***.
- ***Fill Specification:*** dacă această opţiune este expandată, sunt prezentate informaţiile despre ***Fill Factor*** şi ***Pad Index***.
  - ***Fill Factor*** este, de fapt, utilizată pentru a lăsa un spaţiu liber în cadrul paginii, cu scopul de a permite ca modificările sau adăugările ulterioare ale liniilor tabelului să găsească spaţiu într-o pagină. ***Fill factor*** este o valoare între 0 şi 100.

    Valoarea 100 semnifică că pagina va fi plină. Se va necesita cel mai mic spaţiu de memorie. Această opţiune se recomandă să fie utilizată atunci când datele nu vor fi actualizate, de exemplu, în tabele numai pentru citire.

    O valoare mai mică lasă mai mult spaţiu liber în paginile cu date. Această opţiune reduce necesitatea de divizare a paginilor cu date atunci când indecşii cresc, dar necesită mai mult spaţiu de păstrare.

- ***Pad Index*** – *indică dacă, în acest index, paginilor intermediare le sunt oferite acelaşi* procentaj de spaţiu liber (*padding*) specificat în ***Fill Factor***.

- ***Ignore Duplicate Keys*** specifică ce se întâmplă atunci când un rând este inserat pe parcursul operaţiei de inserare în bloc şi valoarea cheii lui este egală cu valoarea cheii existente. Dacă se selectează:
  - ***Yes*** - SQL Server publică un avertisment, ignoră rândul şi încearcă să insereze restul rândurilor.
  - ***No*** - SQL Server publică un mesaj de eroare şi realizează un rollback al operaţiei de inserare în bloc.
- ***Included Columns***: lista de nume ale coloanelor care constituie cheia indexului. Coloanele pentru subchei pot fi specificate numai pentru indecşi de tip *Nonclustered*. Această proprietate este ascunsă pentru indecşii XML.
- ***Is Disabled:*** indică dacă indexul este dezactivat. Aceasta este o proprietate numai pentru citire (read-only). Proprietate ia valoarea ***Yes,*** dacă indexul a fost dezactivat cu instrumente ce nu fac parte din ***Visual Database tools***.
- ***Is Full-Text Key***: specifică dacă indexul este de tip *full-text*. Această proprietate este ascunsă pentru indecşii XML
- ***Page Locks Allowed***: specifică dacă blocarea la nivel de pagină este permisă de acest index. Permiterea sau anularea blocării la nivel de pagină afectează performanţa bazei de date. Se recomandă setarea valorii ***Yes***.
- ***Re-compute Statistics***: - specifică dacă motorul bazei de date în cauză calculează statistici noi atunci când este creat indexul. Recalcularea statisticilor încetineşte procesul de construire a indecşilor, dar, de regulă, îmbunătăţeşte performanţele interogării bazei de date.
- ***Row Locks Allowed:*** specifică dacă blocarea la nivel de rânduri este permisă de acest index. Permiterea sau anularea blocării la nivel de rând afectează performanţa bazei de date. Opţiunea recomandată este ***Yes***.

10. Pentru finalizarea procedurii, este suficient să fie închisă fereastra ***Indexes/Keys*** (se aplică un click pe butonul **Close**). Controlul se va întoarce ferestrei cu structura tabelului şi în această fereastră pentru salvarea modificărilor în structura indexului se apasă butonul **Save** de pe bara de instrumente.

5.3.3. Crearea indecşilor, pornind de la crearea tabelelor

Procedura pentru crearea unui index într-un tabel nou nu se deosebeşte mult de crearea unui index asupra unui tabel existent. Procesul creării indexului este următorul:

1. În arborele ***Object Explorer*** se extinde dosarul bazei de date, unde va fi creat indexul
2. Se dă un click cu butonul drept al mouse-ului pe dosarul ***Tables***.
3. În meniul de context, se face un click pe opţiunea ***New Table***.
4. Se completează structura tabelului.
5. Se face un click pe butonul **Save,** şi se inserează numele tabelului în caseta de dialog ce a apărut, şi apoi se confirmă introducerea (**OK**).
6. Rămâne să se procedeze în acelaşi mod ca în cazul anterior, pornind de la pasul 7.

5.3.4. Modificarea indecşilor

Pentru modificarea unui index, instrumentele visual din ***SQL Server Management Studio*** pot fi utilizate prin două metode.

Prima metodă constă în modificarea indecşilor, utilizând interfaţa de proiectare a tabelelor descrisă în secţiunea *6.3.2.* Din ***Object Explorer*** se accesează tabelul unde se afla indexul necesar şi se modifică proprietăţile lui în fereastra ***Indexes/Keys***. Pentru salvarea setărilor noi ale indexului în fereastra ***Indexes/Keys,*** se aplică un click pe butonul **Close** şi apoi după ce controlul se va întoarce ferestrei cu

structura tabelului, pentru salvarea modificărilor în structura indexului se apasă butonul **Save** de pe bara de instrumente.

Pentru a aplica a doua metodă de modificare a unui index utilizând instrumentele din ***SQL Server Management Studio***, se parcurge următoarea cale:

1. În arborele ***Object Explorer*** se extinde dosarul bazei de date, unde se găseşte indexul necesar
2. Se extinde dosarul ***Tables***.
3. Se extinde (se apasă pe semnul +) nodul tabelului necesar pentru a afişa componentele acestuia.
4. Se extinde dosarul ***Indexes***
5. În lista de indecşi apărută, se alege indexul necesar şi se face un click-drept pe acesta, în meniul de context apărut se alege opţiunea ***Properties***
6. În rezultat, apare fereastra de dialog ***Index Properties.*** Fereastra ***Index Properties*** este separată în două părţi verticale. În partea stângă se alege pagina, în partea dreaptă se află câmpul de lucru, unde se afişează conţinutul acestei pagini. Alegând o pagină sau alta din lista din stânga, se afişează grupul de opţiuni necesare, unde în mod *visual* bifând butoane sau selectând din meniuri derulante valori dezirabile se setează parametrii necesari pentru index. Setările accesibile pentru editare în fereastra ***Index Properties*** sunt similare celor din fereastra ***Indexes/Keys*** din interfaţa ***Table design***.

Fereastra ***Index Properties*** poate fi folosită şi pentru crearea indecşilor noi. În acest caz, după punctul 3 din scenariu de mai sus urmează un click-drept pe dosarul ***Indexes*** şi în meniul de context se alege opţiunea *New Index...*

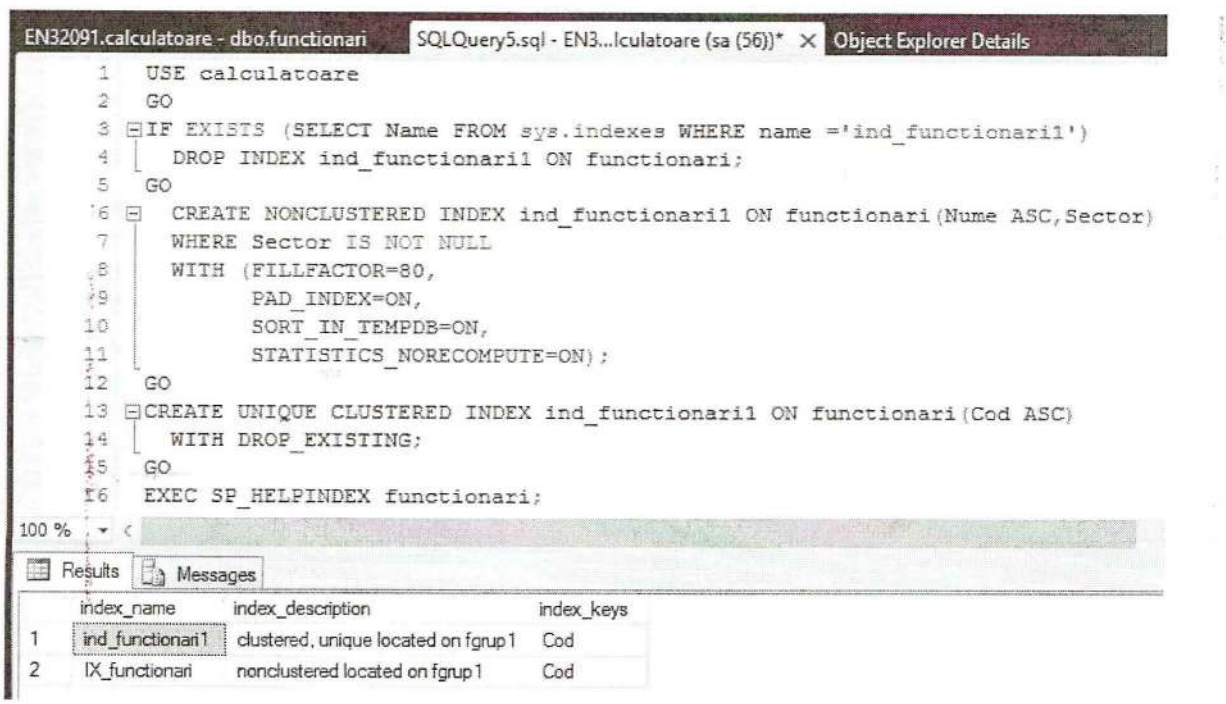
5.4. Gestiunea indecşilor în limbajul Transact-SQL

5.4.1. Crearea indecşilor

Câteva exemple de creare a indecşilor, deja, au fost examinate în procesul creării tabelelor. Se va examina (în figura de mai jos) încă un exemplu de creare a indecşilor cu setări puţin mai avansate.

Instrucţiunea *USE* ***calculatoare*** indică sistemului să utilizeze baza de date ***calculatoare*** pentru sesiunea de lucru curentă.

Instrucţiunea *GO* este un separator de pachete implicit.



Pe a treia linie porneşte instrucţiunea care caută în tabelul de sistem ***indexes*** un index cu acelaşi nume ca şi al celui care se creează şi, dacă îl găseşte, îl şterge. Linia a şasea de cod creează un index ***ind_functionari1*** asupra tabelului ***funcţionari***. Indexul creat este *NONCLUSTERED,* iar *FILTERED* este alcătuit din două coloane ***Nume*** şi ***Sector*** şi are un predicat de filtrare care, în calitate de criteriu, utilizează coloana ***Sector,*** care nu trebuie să fie nulă. Suplimentar, pentru ***ind_functionari1*** sunt definite explicit aşa proprietăţi, precum *FILLFACTOR=80, PAD_INDEX=ON* şi este activat parametrul *STATISTICS_NORECOMPUTE.*

Activarea parametrului *SORT_IN_TEMPDB* indică faptul că, pentru construirea indexului, sistemul va utiliza tabelul ***tempdb***, păstrând în acesta rezultatele intermediare de sortare. Această proprietate este utilă când fişierele bazei de date se află pe discuri fizice diferite.

Linia a treisprezecea creează un index de tip *CLUSTERED* cu numele ***ind_functionari2*** şi cu valori unice pentru câmpul ***Cod*** al tabelului ***functionari***.

La acest index, este adăugată clauza *DROP_EXISTING,* care, în procesul creării acestuia, verifică dacă pentru tabelul respectiv nu mai există un index cu acest nume şi îl şterge în caz afirmativ. Evitând posibilele erori, această clauză este utilă, dacă e nevoie de reconstruirea indexului. Dacă indexul cu astfel de nume încă nu există în sistem, clauza *DROP_EXISTING* va eşua.

5.4.2. Modificarea indecşilor

Câteodată poate apărea necesitatea de reconstruire a indecşilor sau de modificare a proprietăţilor acestora. Mai jos, sunt prezentate câteva exemple de modificare şi reconstruire a indecşilor în limbajul Transact-SQL.

Exemplul de mai jos reconstruieşte indexul ***ind_functionari1*** din tabelul ***functionari***:

```
USE calculatoare;
GO
ALTER INDEX ind_functionari1 ON functionari REBUILD;
GO
```

Instrucţiunea de reconstruire a indexului de mai jos conţine parametrul *ALL,* fapt ce permite reconstruirea tuturor indecşilor asociaţi tabelului cu opţiuni specificate explicit după clauza *REBUILD WITH...*

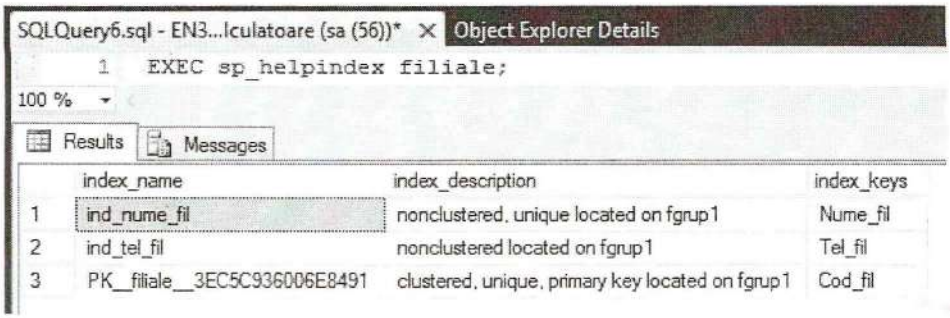
```
USE calculatoare;
GO
ALTER INDEX ALL ON functionari
REBUILD WITH (FILLFACTOR = 70, STATISTICS_NORECOMPUTE=ON);
GO
```

Indexul poate fi dezactivat, dar definiţiile lui să rămână în metadate. Dezactivând un index de tip *Clustered,* se dezactivează accesul către tabelul respectiv. Indexul dezactivat poate fi activat din nou prin reconstruirea indexului. Dacă este nevoie de dezactivat un index, instrucţiunea, în acest caz, va fi similară exemplului de mai jos:

```
USE calculatoare;
GO
ALTER INDEX ind_functionari1 ON functionari DISABLE;
GO
```

5.4.3. Vizualizarea indecşilor

Pentru vizualizarea indecşilor existenţi asociaţi de un tabel, se utilizează procedura stocată **sp_helpindex**. Ea realizează o căutare în tabelul sistemului **sys.indexes**. Această procedură trebuie să fie executată în *Editorul de Interogari Transact-SQL*, precum e prezentat în figura ce urmează:



	index_name	index_description	index_keys
1	ind_nume_fil	nonclustered, unique located on fgrup1	Nume_fil
2	ind_tel_fil	nonclustered located on fgrup1	Tel_fil
3	PK__filiale__3EC5C936006E8491	clustered, unique, primary key located on fgrup1	Cod_fil

Întrebări de control

1. Sintaxa de bază a instrucţiunii *CREATE TABLE.*
2. Sintaxa de bază a instrucţiunii *ALTER TABLE.*
3. Sintaxa de bază a instrucţiunii *INSERT.*
4. Sintaxa de bază a instrucţiunii *DELETE.*
5. Sintaxa de bază a instrucţiunii *UPDATE.*
6. Deosebirea dintre instrucţiunile *DELETE* şi *TRUNCATE TABLE.*
7. Tipuri de indecşi acceptaţi de MS SQL Server 2014.
8. Sintaxa de bază a instrucţiunii *CREATE INDEX.*

Sarcini practice:

1. Să se suprime baza de date ***calculatoare*** creată în lucrările 3-4.
2. Să se creeze baza de date ***calculatoare*** (tabelele, constrângerile de integritate), utilizând limbajul Transact-SQL.
3. Să se creeze tabelul ***imprimante_stoc*** cu aceeaşi structură ca şi a tabelului ***imprimante***. Să se insereze toate datele din tabelul ***imprimante*** în tabelul ***imprimante_stoc***. Să se scrie, cu acest scop, un număr minimal de instrucţiuni.
4. Adăugaţi în tabelul ***produse*** următoarele produse de la producătorul Z: modelul imprimantă - 4003, modelul pc - 4001 şi modelul laptop - 4002. Să se scrie o singură instrucţiune SQL.
5. Adăugaţi în tabelul ***pc_uri*** modelul 4444 cu codul 22, care are viteza procesorului 1200 şi preţul 1350. Caracteristicile care lipsesc trebuie să fie completate cu valori implicite definite pentru coloanele respective. Pentru realizarea sarcinii cu succes, dacă este necesar, înainte de inserarea datelor, să se modifice schema tabelului, utilizând instrucţiunile DDL respective.
6. Pentru fiecare model de laptopuri, să se adauge o înregistrare în tabelul ***pc_uri*** cu următoarele caracteristici:
   - ***Cod***: codul minimal al *laptopului* în grup +30.
   - ***Model***: numărul de model al *laptopului* +100.
   - ***Viteza***: viteza maximală a *laptopului* în grup.
   - ***Ram***: capacitatea maximală a memoriei operative a *laptopului* în grup *2.
   - ***Hd***: capacitatea maximală a discului dur al *laptopului* în grup *2.

- ***Cd***: valoarea implicită.
- ***Pret***: preţul maximal al *laptopului* în grup, micşorat de 1.5 ori.

7. Să se scrie interogări de creare a indecşilor asupra tabelelor din baza de date ***calculatoare*** pentru a asigura o performanţă sporită la executarea interogărilor *SELECT* din Lucrarea practică 4. Rezultatele optimizării sa fie analizate în baza planurilor de execuţie până şi după crearea indecşilor.

6. DIAGRAME, SCHEME ŞI SINONIME

În capitol se tratează subiectele cu privire la crearea diagramelor şi modificarea formelor de vizualizare ale acestora, setarea legăturilor dintre tabele, gestionarea şi editarea obiectelor bazei de date, folosind diagramele. În afară de aceasta, sunt examinate aspectele de creare, modificare şi suprimare a schemelor, de deplasare a obiectelor între scheme, de interogare a schemelor, precum şi de creare şi suprimare a sinonimelor [2].

6.1. Diagramele în baza de date SQL Server

Diagramele bazei de date reprezintă tabelele din baza de date în mod grafic. Diagramele afişează denumirile coloanelor conţinute în tabele, legăturile dintre tabele, indecşii şi constrângerile asociate tabelelor.

Diagramele bazei de date pot fi folosite pentru:

- Vizualizarea structurii tabelelor bazei de date şi a legăturilor dintre tabele.
- Modificarea structurii bazei de date.
- Crearea tabelelor noi, indecşilor, referinţelor şi altor constrângeri.
- Furnizarea diferitelor vizualizări ale bazelor de date complexe.
- Manipularea obiectelor bazei de date, fără a scrie coduri în Transact-SQL.

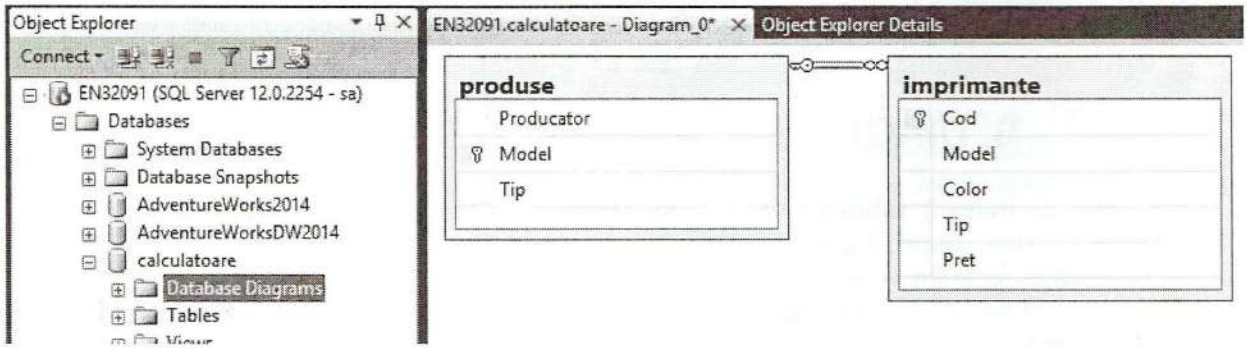
6.1.1. Crearea diagramelor

Pentru crearea diagramei unei baze de date, se execută următorii paşi:

1. Se iniţializează SQL Server.
2. Se extinde un grup de servere (dacă există grupare) şi apoi se extinde un server.
3. Se extinde dosarul ***Databases*** şi se extinde baza de date pentru care se doreşte construirea unei diagrame.
4. Se face un click cu butonul drept al mouse-ului pe dosarul ***Database Diagrams***.
3. În meniul de context, care s-a deschis, se selectează opţiunea ***New Database Diagram***. În calitate de exemplu, este folosită baza de date ***calculatoare*** creată în laboratoarele precedente.
6. În fereastra apărută, intitulată ***Add Table***, sunt selectate tabelele care trebuie să facă parte din diagrama nouă. În câmpul de lucru al acestei ferestre, este afişată lista tabelelor disponibile pentru crearea diagramei. Tabelele destinate să intre printre cele ce vor constitui diagrama sunt selectate câte unul, când se dă un click asupra butonului **Add.** În mod automat, tabelul este transportat în aria de lucru a diagramei. Când sunt mai multe tabele, pot fi manipulate în grup, utilizând butoanele **Shift** şi **Ctrl**. În cazul în care în diagramă sunt incluse toate tabelele necesare, se face un click pe butonul **Close**.

Se adăugă tabelele ***imprimante*** şi ***produse***. În urma acestor acţiuni, pe ecranul calculatorului, va fi prezentată următoarea diagramă alcătuită din 2 tabele:

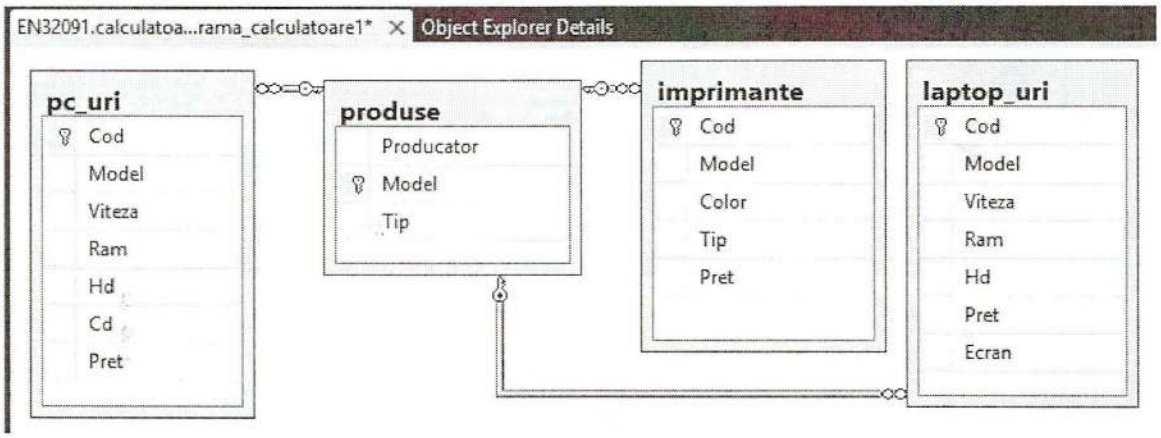
Să se observe că tabelele din diagramă, deja, sunt legate între ele prin anumite conexiuni. Odată construită, diagrama trebuie salvată. Pentru aceasta, se face un click pe butonul **Save,** , urmat de introducerea numelui diagramei în caseta care a apărut. În cazul nostru, se va insera numele ***Diagrama_calculatoare1***.



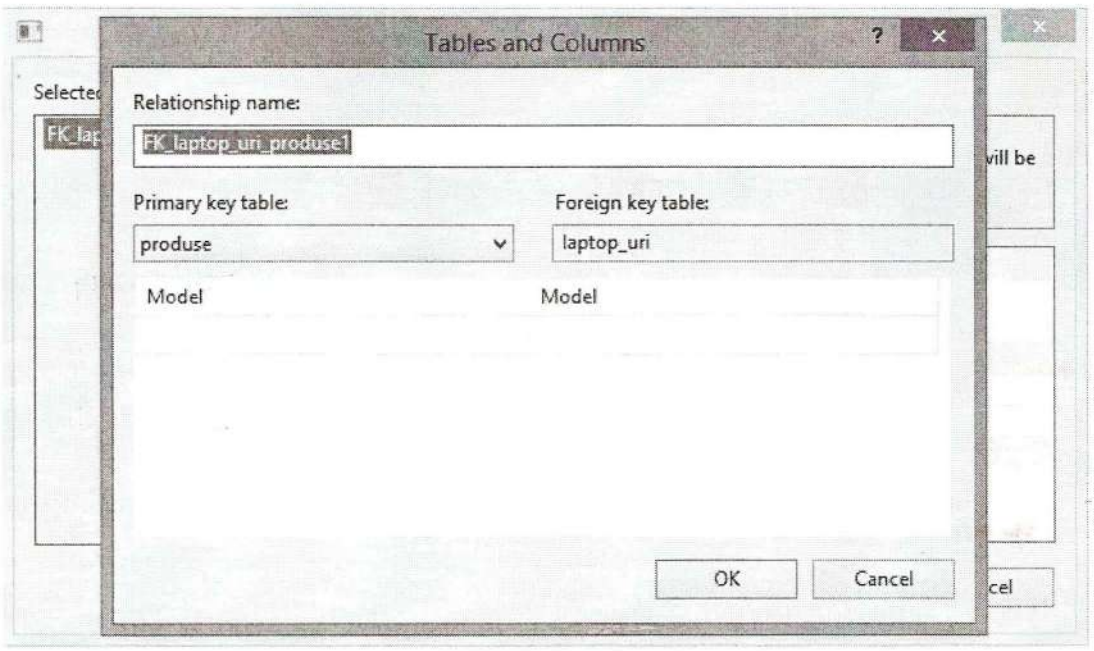
6.1.2. Editarea diagramelor

Diagramele create pot fi modificate, adică obiectele din ele pot fi editate sau şterse în mod manual. Aici, iarăşi, vor fi folosite tabelele create în capitolul 3. Pentru editarea diagramei ***Diagrama_calculatoare1***, se procedează în următorul mod:

1. Se extinde dosarul Databases -> calculatoare -> Database Diagrams.
2. Se face un dublu click pe pictograma ***Diagrama_calculatoare1***. În consecinţă, în câmpul de lucru, se extinde fila cu diagrama respectivă.



3. Se face un click cu butonul drept al mouse-ului asupra oricărei părţi libere a ariei de lucru şi, în meniul de context, se selectează opţiunea ***Add Table.***
4. În fereastra apărută, deja cunoscută, intitulată ***Add Table,*** se selectează cele 2 tabele din schema bazei de date rămase neutilizate.
5. Apoi rămâne doar să se aplice un click pe butonul ***Add***. Diagrama obţinută va fi similară celei de mai sus.
6. Se poate observa, că legăturile dintre unele tabele deja există. Acest lucru se explică prin faptul că la crearea schemei bazei de date deja erau definite constrângerile respective. Dacă constrângerile referenţiale nu sunt definite din start în schema bazei de date, acest lucru poate fi realizat, apelând la interfaţa de creare a diagramelor. Astfel, pentru editarea sau crearea unei legături, fie între tabelul ***produse*** şi ***laptop_uri***, este de ajuns să se aplice un click pe coloana care se găseşte la stânga numelui câmpului unui tabel (***produse***) şi să se gliseze până la poziţia respectivă din tabelul (***laptop_uri***) cu care se va lega. Odată cu finalizarea procesului de glisare, pe ecranul calculatorului apare următoarea fereastră:



7. În această fereastră, SGBD-ul cere confirmarea legăturii. Este posibilă modificarea tabelelor – surselor cheilor primare şi schimbarea atributelor de referinţă. Apoi se apasă butonul **OK**.
8. Apoi devine accesibilă fereastra ***Foreign key relationship,*** în care se editează proprietăţile extinse ale legăturii nou-create. O legătură include următoarele proprietăţi:

General

Expandaţi pentru a prezenta ***Check Existing Data on Creation or RE-Enabling*** şi ***Tables and Columns Specifications***

- ***Check Existing Data on Creation or Re-Enabling:*** verifică datele din tabel înainte ca constrângerea să fie creată sau reactivată.
- ***Tables and Columns Specifications Category***

  Expandaţi pentru a prezenta care coloane din care tabele joacă rolul de cheie externă sau primară (sau unică) în legătură. Pentru a modifica sau defini aceste valori, se aplică un click pe butonul din câmpul respectiv, ...

  - ***Foreign Key Base Table:*** arată care tabel conţine coloana ce joacă rol de cheie externă în legătura selectată.
  - ***Foreign Key Columns:*** arată care coloană joacă rol de cheie externă în legătura selectată.
  - ***Primary/Unique Key Base Table:*** arată care tabel conţine coloana ce joacă rol de cheie primară (sau unică) în legătura selectată
  - ***Primary/Unique Key Columns:*** arată care coloană joacă rol de cheie primară (sau unică) în legătura selectată.

Database Designer

- ***Enforce For Replication:*** indică dacă se aplică constrângerea în cazul când agentul de replicare execută o inserare, modificare sau ştergere în tabelul dat. Pentru informaţii suplimentare, vezi ***Controlling Constraints, Identities*** şi ***Triggers*** cu *NOT FOR REPLICATION.*
- ***Enforce Foreign Key Constraint:*** se specifică dacă este sau nu permisă modificarea datelor în coloanele legăturii, în cazul în care aceste modificări invalidează integritatea constrângerii referenţiale. Să se aleagă opţiunea **Yes,** dacă nu se doreşte astfel de modificări şi **No,** în caz contrar. În cazul nostru, se alege **Yes**.

- ***INSERT and UPDATE Specification***

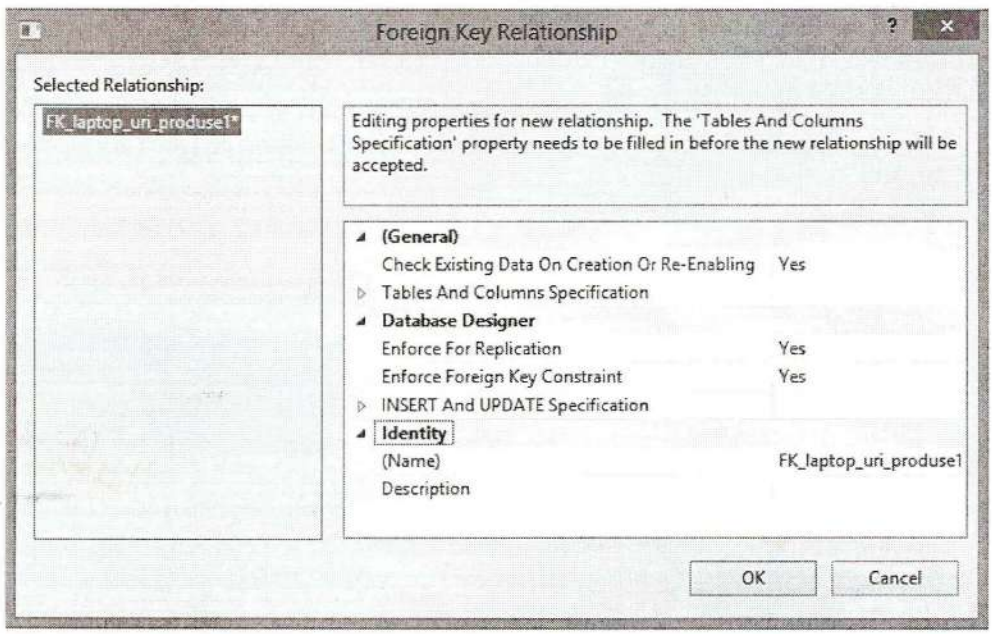
  Expandaţi pentru a vizualiza informaţia despre ***Delete Rule*** şi ***Update Rule*** a legăturii.

  - ***Delete Rule:*** se specifică ce se întâmplă, dacă un utilizator încearcă să şteargă un rând de date implicat într-o legătură de tip cheie externă.
    - ***No Action***: are loc un *roll-back* şi apare un mesaj de eroare care comunică utilizatorului că ştergerea nu este permisă.
    - ***Cascade:*** se şterg toate rândurile ce conţin date implicate în legătura cheii externe.
    - ***Set Null:*** se setează valoarea *NULL*, dacă toate coloanele cheii externe din tabel pot accepta această valoare.
    - ***Set Default:*** se setează valoarea implicită, dacă toate coloanele cheii secundare din tabel au definite valori implicite.
  - ***Update Rule:*** se specifică ce se întâmplă, dacă un utilizator încearcă să modifice un rând de date implicat într-o legătură de tip cheie externă.
    - ***No Action:*** are loc un *roll-back* şi apare un mesaj de eroare care comunică utilizatorului că ştergerea nu este permisă.
    - ***Cascade:*** se şterg toate rândurile ce conţin date implicate în legătura cheii externe.
    - ***Set Null:*** se setează valoarea *NULL*, dacă toate coloanele cheii externe din tabel pot accepta această valoare.
    - ***Set Default:*** se setează valoarea implicită, dacă toate coloanele cheii secundare din tabel au definite valori implicite. Se aplică începând cu ***SQL Server 2005***.

Identity

Expandaţi pentru a afişa proprietăţile câmpurilor ***Name*** şi ***Description***

- ***Name:*** indică numele legăturii. Dacă o nouă legătură este creată, se asociază un nume implicit bazat pe tabelul din fereastra activă ***Table Designer.*** Numele poate fi modificat.
- ***Description:*** descrie legătura. Pentru o descriere mai detaliată, se aplică un click pe ***Description*** şi apoi un click pe butonul ... Aceasta oferă o suprafaţă pentru inserarea textului.

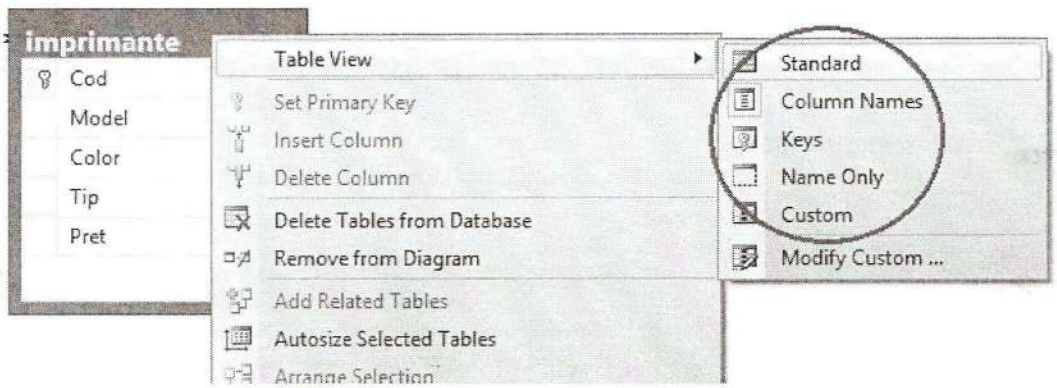


9. După setarea proprietăților legăturilor, se apasă **OK**. În caz concret, tabelul ***produse*** este legat cu tabelul ***laptop_uri***, ***imprimante*** şi ***pc_uri*** (legătură de tipul 1:m). Rareori, doar în practică sunt cazuri când, în 2 tabele diferite, coincid cheile primare, atunci legătura este de tipul 1:1.
10. După modificările executate, pentru a le salva se apasă butonul **Save**. Ca urmare, apare fereastra care ne informează despre faptul că schimbările operate în schema bazei de date vor fi salvate.

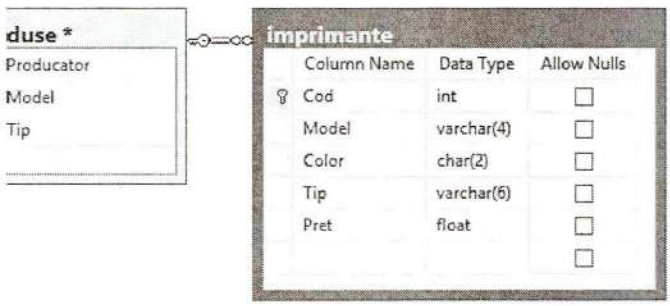
6.1.3. Modificarea formei de vizualizare a tabelelor

Diagrama bazei de date afişează tabelele în formă de dreptunghiuri, în care sunt incluse denumirile de coloane ale tabelelor. Apelând la meniul de context, formatul de prezentare a tabelului poate fi modificat. Pentru a realiza această sarcină, se procedează precum urmează:

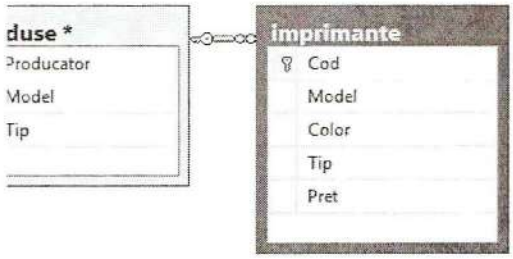
1. Se aplică un click cu butonul drept al mouse-ului asupra barei de titlu a tabelului.
2. În meniul de context, se selectează opțiunea ***Table View***.
3. Apoi se trece la opțiunile meniului suspendat şi se acționează, în funcție de necesități, una din ele. Există 5 opțiuni de afişare a tabelelor disponibile.



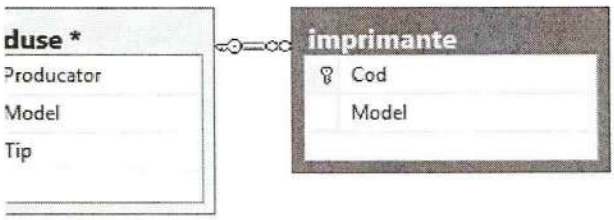
- Opțiunea ***Standard*** prezintă un grid cu toate coloanele tabelului şi proprietățile lor. În figura ce urmează, opțiunea ***Standard*** este aplicată asupra tabelului ***imprimante***:



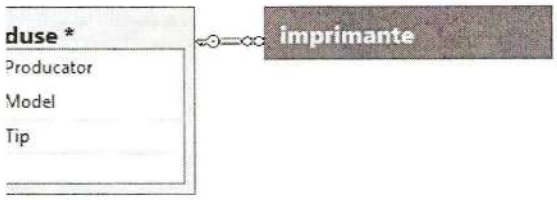
- Opțiunea ***Column Names*** prezintă numai numele tuturor coloanelor. Atunci când un tabel existent este adăugat la diagramă, el apare în acest format:



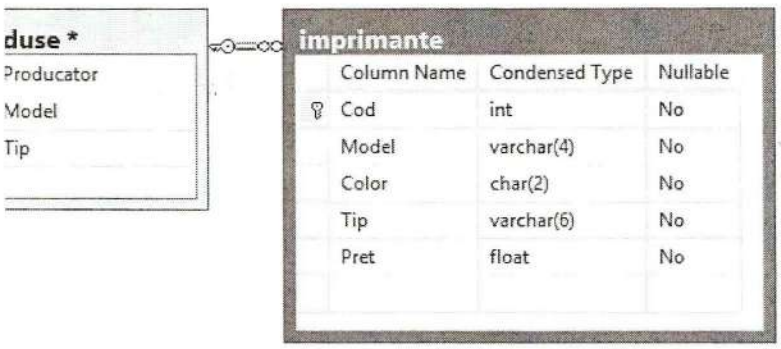
- Opţiunea ***Keys*** prezintă un grid, ce conţine numele tuturor coloanelor, care fac parte din cheia primară, cheile externe sau care au caracteristica *UNIQUE*:



- Opţiunea ***Name Only*** prezintă numai bara de titlu cu numele tabelului. În figura de mai jos, tabelul ***carti*** este prezentat în urma acţiunii opţiunii ***Name Only***:



- Opţiunea ***Custom*** prezintă coloanele tabelului numai cu proprietăţile definite de utilizator. În figura ce urmează, sunt prezentate doar proprietăţile ***Column Name, Condensed Type, Nullable***:



Trebuie menţionat că proprietăţile care vor fi afişate în forma de prezentare ***Custom*** pot fi definite în acelaşi meniu, opţiunea ***Modify Custom***..

6.2. Administrarea schemelor

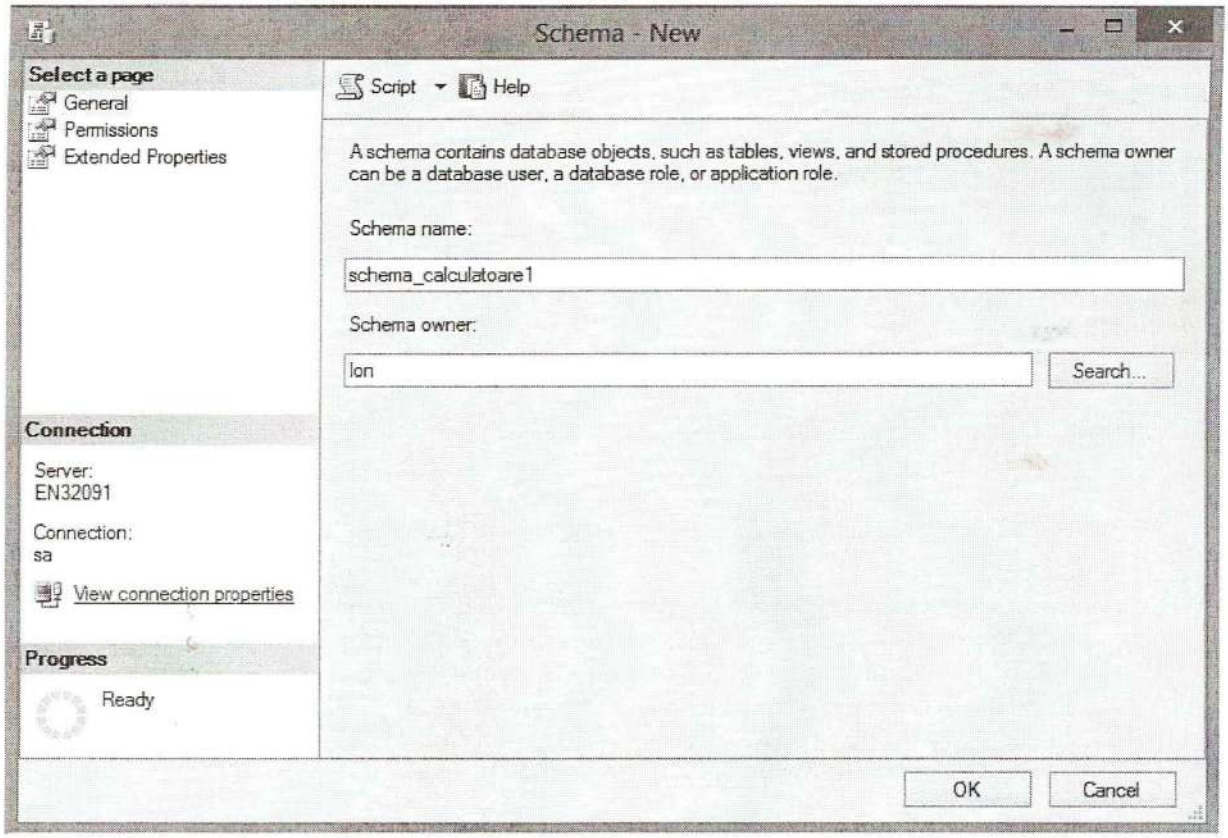
Utilizarea schemelor oferă mai multe avantaje pentru simplificarea procesului de gestiune a datelor şi permite crearea submulţimilor de obiecte care pot fi administrate ca un tot întreg. Schemele pot fi utilizate pentru limitarea obiectelor vizibile (accesibile) pentru un utilizator sau un grup de utilizatori, fapt ce permite o partajare mai simplă a accesului la baza de date. Câţiva utilizatori pot deţine o schema fiind membri ai unui grup de utilizatori din Windows. O schemă poate fi definită una implicită pentru unu sau mai mulţi utilizatori.

Schemele sunt nişte containere de obiecte utilizate pentru definirea spaţiilor de nume (*namespaces*) ale obiectelor bazei de date. Schemele se află în corpul unei baze de date. O bază de date poate avea mai multe scheme. O schemă poate conţine un număr de obiecte limitat de numărul de obiecte ce poate maximal susţine o bază de date gestionată de SQL Server. Ca rezultat, se formează o ierarhie logică de organizare a datelor: ***Baza de Date -> Schema -> Obiectele schemei*** (*Tabele, Viziuni* etc.). O schema nu poate conţine în corpul său alte scheme. O schemă nu poate conţine în corpul său obiecte cu nume duplicat.

6.2.1. Crearea schemelor în mediul SQL Server Management Studio

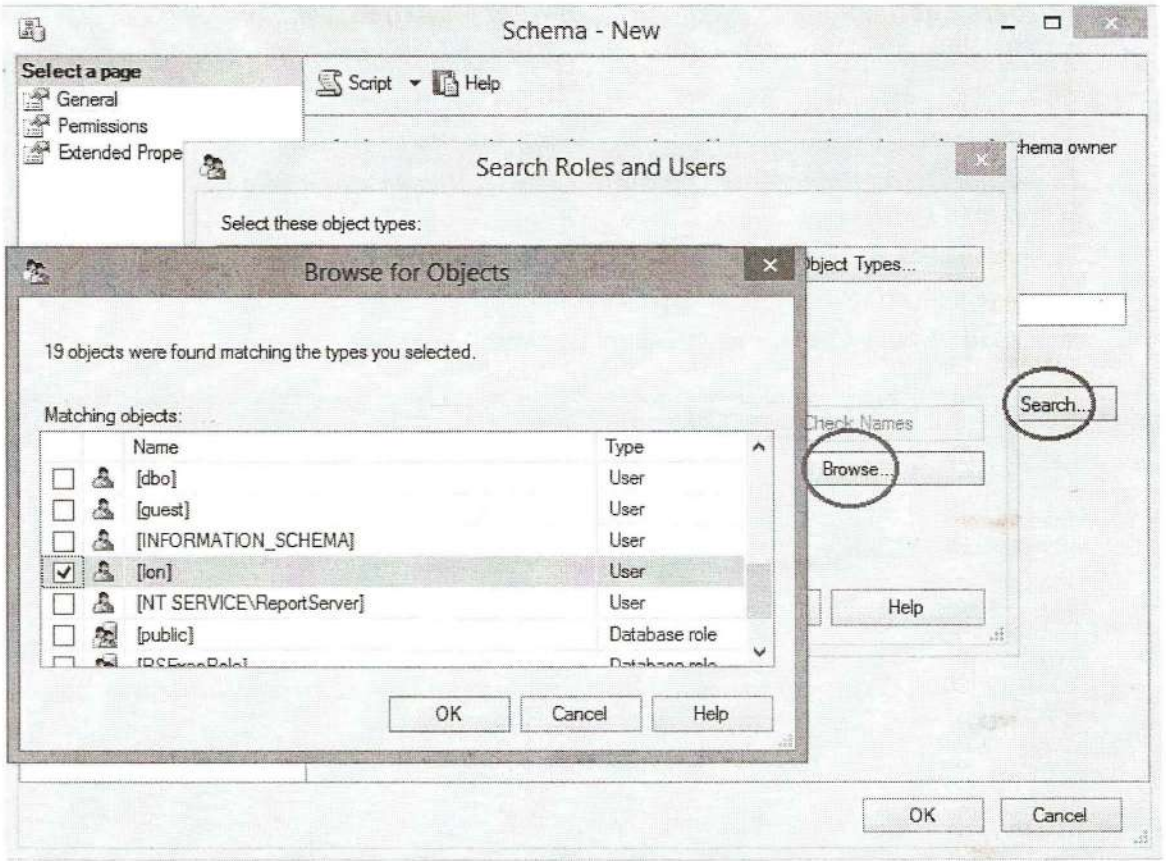
Pentru crearea schemei noi cu ajutorul ***SQL Server Management Studio***, trebuie să fie executaţi următorii paşi:

1. Se conectează serverul unde se află baza de date necesară.
2. Pe panelul ***Object Explorer*** se expandează dosarul ***Databases*** (se face un click pe semnul „+" al dosarului ***Databases***).
3. Se face un click pe semnul „+" lângă baza de date necesară.
4. În arborele de obiecte apărut, se expandează dosarul ***Security.***
5. În dosarul ***Security,*** se face un clic-drept pe dosarul ***Schemas.***



6. În meniul de context apărut, se alege opţiunea ***New Schema...***

   În acest punct, pe ecranul calculatorului, va apărea fereastra de dialog (***Schema - New***), unde se vor defini proprietăţile schemei preconizate:
7. Pe pagina ***General*** în câmpul ***Schema name*** se scrie numele schemei, iar în câmpul ***Schema owner*** se defineşte proprietarul acesteia.
8. Pentru căutarea conturilor utilizatorilor care pot primi rolul de proprietari ai schemei, se apasă butonul **Search**. Va apărea o fereastră de dialog ***Search Roles and Users*** şi în această fereastră se apasă butonul **Browse** pentru a deschide fereastra următoare ***Browse for Objectcts***.
9. În fereastra ***Browse for Objectcts,*** se alege un *rol* sau un *utilizator* care va fi proprietarul schemei, apoi se confirmă obiectul ales prin apăsarea butonului **OK.**



10. După ce numele schemei şi proprietarul acesteia este definit, se apasă butonul **OK** şi schema va fi creată.

    Proprietar al schemei poate deveni orice *utilizator*, *rol* sau *aplicaţie* care are acces la baza de date. Un utilizator poate deţine mai multe scheme, însă numai o schemă poate fi implicită. Dacă în calitate de proprietar al schemei este definit un rol sau un grup Windows, atunci schema poate fi deţinută paralel de câţiva utilizatori.

6.2.2. Crearea schemelor în limbajul Transact-SQL

Atunci când se creează o schemă, folosind instrucţiunile Transact-SQL, aceasta se va găsi în baza de date curentă. Pentru crearea schemelor prin cod, se utilizează instrucţiunea *CREATE SCHEMA*. Sintaxa de bază a instrucţiunii este:

```
CREATE SCHEMA clauza_nume_schema [ <element_schema> [ ...n ] ]

< clauza_nume_schema> ::=
        { nume_schema
        | AUTHORIZATION nume_detinator
        | nume_schema AUTHORIZATION nume_detinator }

< element_schema > ::=
        {   definiţie_tabel | definiţie_viziune | acordare_drept
            revocare_drept | interzicere_drept}
```

Precum se vede, sintaxa instrucţiunii de creare a schemei în afară de numele şi proprietarul acesteia, poate include clauze de creare a tabelelor sau viziunilor, de definire a permiselor de acces pentru un utilizator sau un grup de utilizatori.

Următorul exemplu creează o schemă cu numele ***schema_calculatoare2*** şi proprietarul schemei *Ion*. Schema mai conţine instrucţiunea de creare a tabelului ***vinzari***. Tot aici, utilizatorului *Anna* îi este acordată permisiunea de selectare a datelor din această schemă, iar utilizatorului *Vasile* i se interzice această operaţiune. Atât schema, cât şi tabelul sunt create în cadrul unei singure instrucţiuni.

```
USE calculatoare;
GO
CREATE SCHEMA schema_calculatoare2 AUTHORIZATION Ion
    CREATE TABLE vinzari(Comanda INT, Model VARCHAR(4), Cod_client INT,
                PRIMARY KEY(Comanda))
    GRANT SELECT ON SCHEMA:: schema_calculatoare2 TO Anna
    DENY  SELECT ON SCHEMA:: schema_calculatoare2 TO Vasile
GO
```

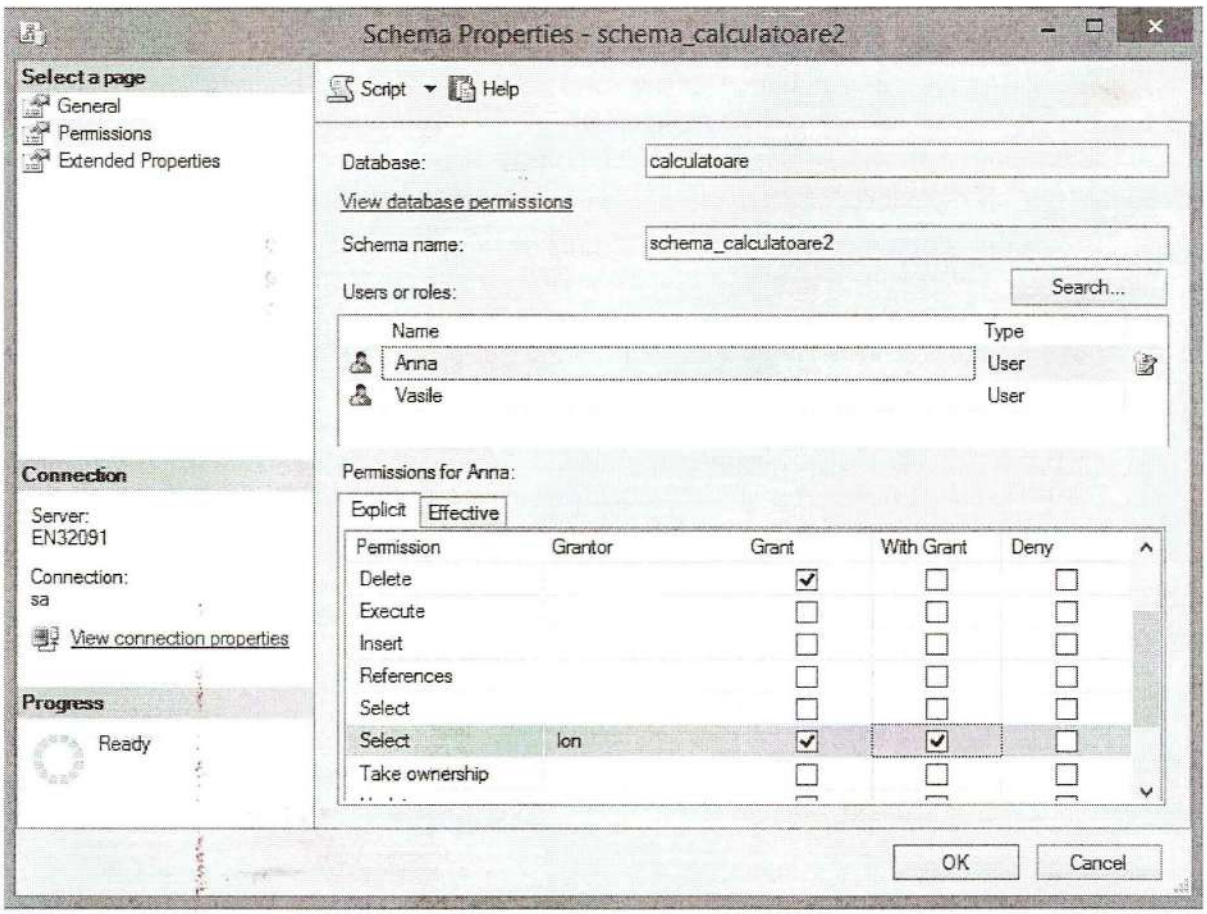
6.2.3. Modificarea schemelor

Periodic, poate apărea necesitatea de oferire a schemei unui proprietar nou, de modificare a permiselor de acces unor sau altor utilizatori.

Pentru modificarea proprietarului schemei cu ajutorul ***SQL Server Management Studio***, se execută următoarea consecutivitate de operaţiuni:

1. Se parcurge calea 1-5 menţionată în secţiunea anterioară (*Crearea schemelor*).
2. Se expandează dosarul ***Schemas*** şi se face un click-drept pe schema care trebuie modificată.
3. Din meniul de context apărut se alege ***Properties***.

   În rezultat apare fereastra ***Schema properties*** cu interfaţa deja cunoscută. Pentru a modifica proprietarul schemei, se execută o serie de paşi identici operaţiunilor de definire a proprietarului descrişi anterior.



Pentru setarea mai detaliată a accesului la schemă, se execută următoarea serie de paşi:

1. În partea dreaptă a ferestrei ***Schema properties*** în lista ***Select a page,*** se alege pagina ***Permissions***

2. Pentru adăugarea accesului explicit unui utilizator sau rol, se apasă butonul ***Search***. În consecinţă, se va deschide fereastra de dialog ***Select Users and Roles***.
3. Se inserează numele utilizatorului sau al rolului. Numele se separă prin punct şi virgulă. Pot fi introduse şi nume incomplete, apoi se face un click pe butonul **Check Names** pentru completarea acestora. După aceasta, se apasă **OK**.
4. În rezultat, utilizatorii apar în tabelul ***Users or roles***. Selectând utilizatorul respectiv în tabelul ***Users or roles,*** pot fi editate permisiunile de acces detaliate în tabelul de mai jos ***Permissions for ...***
5. După executarea modificărilor, se apasă **OK**.

Trebuie menţionat că schema nu poate fi redenumită. Drept soluţie, în acest caz, se creează o schemă nouă cu un alt nume.

6.2.4. Deplasarea obiectelor în scheme cu ajutorul SQL Management Studio

Deoarece schemele sunt nişte containere de obiecte, există posibilitatea deplasării obiectelor dintr-o schemă în alta. Deplasarea obiectelor este posibilă numai în cadrul unei baze de date. Permutarea obiectelor modifică şi permisiunile de acces ale utilizatorilor la aceste obiecte. Toate drepturile de acces anterioare sunt suprimate. Dacă proprietarul obiectului deplasat este un utilizator sau un rol definite explicit, atunci aceştia şi în continuare vor fi proprietarii acestui obiect.

Pentru deplasarea unui obiect într-o schemă nouă cu ajutorul ***SQL Management Studio,*** se parcurge următoarea cale:

1. Conectarea la serverul unde se află baza de date necesară.
2. Pe panelul ***Object Explorer,*** se expandează dosarul ***Databases.***
3. Se face un click pe semnul „+" lângă baza de date respectivă, pentru a afişa arborele de obiecte.
4. În arborele de obiecte, se selectează obiectul căutat, fie un tabel sau o viziune.
5. Se face un click- drept pe acest obiect şi în meniul de context se alege ***Design***.
6. Pe panelul ***Properties,*** în secţiunea ***Identity*** se alege meniul derulant ***Schema***, unde se selectează schema nouă. Dacă panelul ***Properties*** nu este afişat în câmpul de lucru, el poate fi chemat, apăsând tasta **F4**.
7. După modificările necesare, se apasă butonul **Save** de pe bara de instrumente.

   La deplasarea obiectului în altă schemă, trebuie să se ţină cont de alte obiecte cu care acesta se află în dependenţă. Pentru vizionarea dependenţelor obiectului, în ***Object Explorer*** se face un click-drept pe obiectul respectiv şi din meniul de context se alege ***View Dependencies.*** În fereastra apărută ***Object Dependencies,*** sunt vizualizate toate legăturile obiectului respectiv.

6.2.5. Deplasarea obiectelor cu ajutorul limbajului Transact-SQL

Pentru transferarea obiectelor cu ajutorul limbajului Transact-SQL, se execută instrucţiunea *ALTER SCHEMA*. Sintaxa generală a instrucţiunii este următoarea:

```
ALTER SCHEMA schema_destinatie
        TRANSFER schema_sursa.nume_obiect_deplasat
```

Mai jos, este prezentat un exemplu de instrucţiune, unde tabelul ***vinzari*** este transferat din schema ***schema_calculatoare2*** în ***schema_calculatoare1***.

```
USE calculatoare;
GO
ALTER SCHEMA schema_calculatoare1
                TRANSFER schema_calculatoare2.vinzari;
GO
```

6.2.6. Interogarea schemelor

La crearea unui utilizator, acestuia i se asociază o schemă implicită, care este utilizată de server pentru procesarea numelor de obiecte definite incomplet. La adresarea către obiectele din cadrul schemei implicite, numele schemei poate fi omis. Pentru a se adresa obiectelor din altă schemă, este necesară utilizarea unui identificator constituit din 2 sau 3 părţi. În SQL Server schema implicită pentru utilizatorii noi este ***dbo***. În continuare, sunt prezentate exemple de utilizare a instrucţiunii *SELECT,* care utilizează două scheme.

Interogarea ce urmează afişează modelele, preţurile modelelor şi clienţii care au procurat aceste modele. Tabelele interogate se află în scheme diferite.

```
SELECT schema_calculatoare1.vinzari.Model,
       schema_calculatoare1.vinzari.Cod_client,
       dbo.pc_uri.Pret
     FROM schema_calculatoare1.vinzari, dbo.pc_uri
     WHERE dbo.pc_uri.Model=schema_calculatoare1.vinzari.Model
```

Interogarea de mai jos este similară celei anterioare. Deoarece interogarea este scrisă din numele utilizatorului *Ion*, pentru care schema ***schema_calculatoare1*** este implicită, obiectele ce aparţin acestei scheme pot fi redate în formă prescurtată.

```
SELECT vinzari.Model,
       Cod_client,
       dbo.pc_uri.Pret
     FROM vinzari, dbo.pc_uri
     WHERE dbo.pc_uri.Model=vinzari.Model
```

Într-o bază de date SQL Server, pot exista tabele sau alte obiecte cu acelaşi nume cu condiţia că acestea aparţin unor scheme diferite. În cazul dat, adresarea către aceste tabele trebuie să fie în mod explicit pentru a evita ambiguitatea în tratarea interogărilor.

6.2.7. Ştergerea schemelor

Înainte de a şterge o schemă, este important să fie deplasate sau şterse obiectele incluse. Tentativa de a şterge o schemă care conţine obiecte se va termina cu eşec.

Pentru a şterge o schemă în interfaţa ***SQL Management Studio***, se parcurge următoarea cale:

1. În arborele de obiecte ***Object Explorer*** se face un click-drept pe pictograma schemei care trebuie ştearsă.
2. În meniul de context apărut, se alege opţiunea ***Delete***.
3. În rezultat, apare o fereastră de dialog ***Delete Object***, unde pentru a confirma ştergerea schemei se apasă butonul **OK**.

Instrucţiunea Transact-SQL de ştergere a unei scheme are următoarea formă generala:

```
DROP SCHEMA <nume_schema>
```

6.3. Crearea şi suprimarea sinonimelor

6.3.1. Ce este un sinonim?

Un sinonim este un obiect al bazei de date care serveşte pentru următoarele scopuri:

- Oferă un nume alternativ pentru un obiect din baza de date. La sinonim care se face referire ca la numele unui obiect de bază, care poate exista pe un server local sau pe un server îndepărtat.
- Oferă un nivel de abstractizare care protejează o aplicaţie-client de modificările ce pot fi făcute asupra numelor obiectelor de bază sau locaţiilor acestora.

Sinonimele pot fi utilizate în interogări sau proceduri stocate în loc de numele originale ale obiectelor compuse din 2, 3 sau 4 părți. De exemplu, este nevoie să se execute o interogare către un obiect stocat pe alt server. Atunci pentru accesarea acestui obiect, este necesar să se scrie numele lui explicit ***Nume_server.Nume_baza_de_date.Nume_schema.Nume_Obiect***. Utilizând un sinonim creat în schema curentă, obiectul poate fi accesat doar specificând sinonimul respectiv.

Un sinonim aparține unei scheme, ca şi alte obiecte ale schemei. Numele unui sinonim trebuie să fie unic. Un sinonim poate fi creat pentru următoarele obiecte din baza de date: *tabelele* definite de utilizator (inclusiv tabelele temporare atât locale, cât şi globale), *viziuni*, *proceduri stocate*, *funcții*.

Un sinonim nu poate fi obiect de bază pentru un alt sinonim, şi un sinonim nu poate referi o funcție de agregare definită de utilizator.

Doar numele unui sinonim execută legătura între acesta şi obiectul lui de bază. Existența, tipul şi verificarea drepturilor de acces către obiectul de bază sunt amânate până la rulare. Prin urmare, obiectul de bază poate fi modificat, suprimat sau şters şi înlocuit cu un alt obiect, care are acelaşi nume ca şi obiectul de bază original.

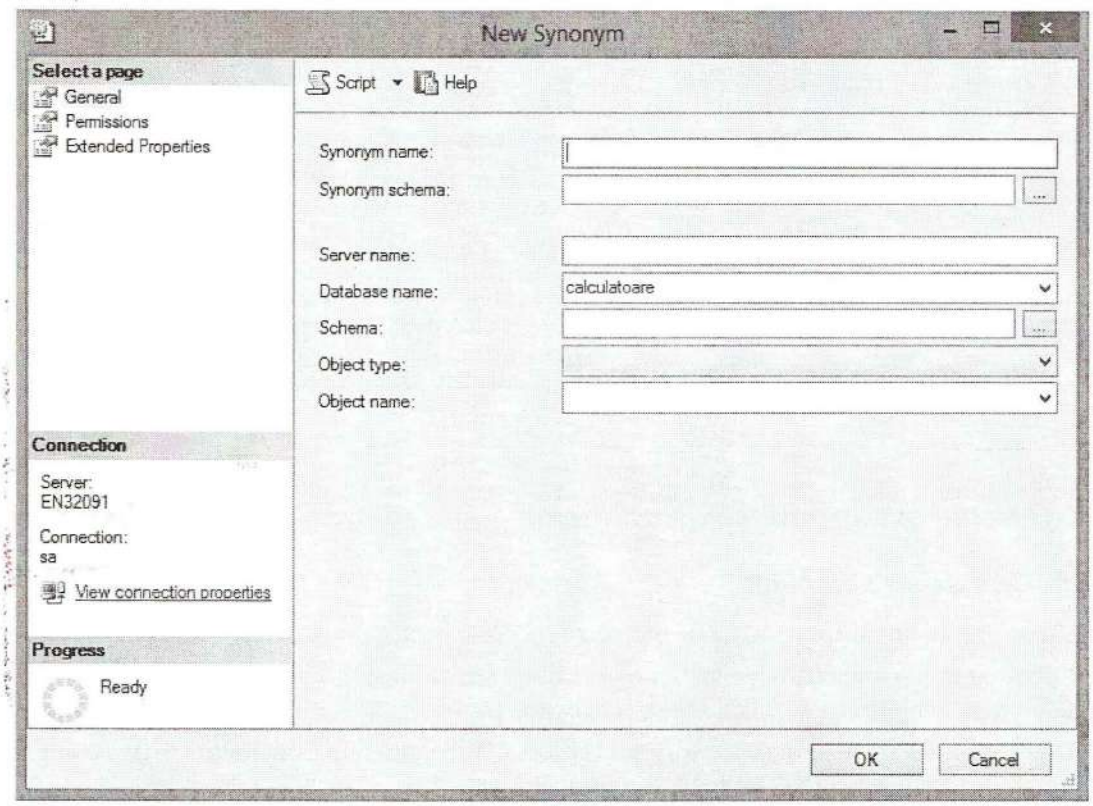
De exemplu, se consideră un sinonim, ***cumparaturi***, care face referire la ***dbo.vinzari*** în baza de date ***calculatoare***. Dacă tabelul ***dbo.vinzari*** este şters şi înlocuit cu o viziune numită ***dbo.vinzari***, atunci ***cumparaturi*** acum se referă la viziunea ***dbo.vinzari***.

6.3.2. Crearea sinonimelor în SQL Management Studio

Pentru crearea sinonimelor, se execută următorii paşi:

1. Conectarea la serverul unde se află baza de date necesară.
2. Pe panelul ***Object Explorer***, se expandează dosarul ***Databases.***
3. Se expandează baza de date necesară.
4. Se face un click-drept pe dosarul ***Synonyms;***
5. În meniul de context apărut se alege opțiunea ***New Synonym...***

   În acest punct, pe ecranul calculatorului, va fi prezentată o fereastră de dialog ***New Synonym*** similară celei de mai jos:



În fereastra *New Synonym* pe pagina ***General,*** se completează următoarele câmpuri:

- ***Synonym name*** - se scrie numele sinonimului.
- ***Synonym schema*** – se scrie schema în care sinonimul nou va fi încorporat.
- ***Server name*** – se scrie numele instanței de server respectiv
- ***Database name*** – se scrie sau se alege din meniul derulant baza de date ce conține obiectul.
- ***Schema*** – se scrie sau se selectează din meniul derulant schema ce conține obiectul de bază.
- ***Object type*** – se selectează din meniul derulant tipul obiectului.
- ***Object name*** – se scrie sau se alege din meniul derulant obiectul de bază la care se referă sinonimul.

Dacă câmpurile ***Synonym schema*** şi ***Server name*** rămân nepopulate, atunci, pentru utilizatorul curent, sistemul ia în considerație parametrii impliciți.

6. După ce toate modificările necesare sunt setate, se apasă butonul **OK**.

6.3.3. Crearea sinonimelor în limbajul Transct-SQL

Pentru crearea unui sinonim în limbajul Transct-SQL, se utilizează următoarea sintaxă generală:

```
CREATE SYNONYM [ nume_schema_1. ] nume_sinonim FOR < obiect >
< obiect > :: =
{
[ nume_server.[nume_baza_de_date].[nume_schema_2].nume_obiect|
 nume_baza_de_date. [nume_schema_2].|
 nume_schema_2.] nume_obiect
}
```

- ***nume_schema_1*** – specifică schema pentru care acest sinonim a fost creat. Dacă schema nu este specificată, atunci sistemul utilizează schema implicită a utilizatorului curent.
- ***nume_sinonim*** - este numele sinonimului nou.
- ***nume_server*** - este numele serverului pe care este stocat obiectul de bază.
- ***nume_daza_de_date*** - e numele bazei de date, în care se află obiectul de bază. Dacă *nume_daza_de_date* nu este specificat, atunci este utilizat numele bazei de date curente.
- ***nume_schema_2*** - este numele schemei obiectului de bază. Dacă numele schemei nu este specificat, atunci este utilizat numele schemei implicite a utilizatorului curent
- ***nume_obiect*** - este numele obiectului de bază la care se referă sinonimul creat.

Următorul exemplu, mai întâi, creează un sinonim ***notebook*** pentru obiectul de bază, ***laptop_uri*** din baza de date ***calculatoare***, apoi interoghează baza de date, utilizând sinonimul creat.

```
USE calculatoare;
GO
CREATE SYNONYM notebook FOR calculatoare.dbo.laptop_uri;
GO
SELECT * FROM notebook;
GO
```

6.3.4. Ştergerea sinonimelor

Pentru a şterge un sinonim, folosind interfața ***SQL Management Studio***, se parcurge următoarea cale:

1. Conectarea serverului, unde se află baza de date necesară.

2. Pe panelul ***Object Explorer,*** se expandează dosarul ***Databases.***
3. Se expandează baza de date necesară.
4. Se expandează dosarul ***Synonyms.***
5. Se face un click-drept pe pictograma sinonimului preconizat spre ştergere, în meniul de context apărut se alege opţiunea ***Delete***.
6. În rezultat, apare o fereastră de dialog deja cunoscută - ***Delete Object,*** unde pentru a confirma ştergerea schemei se apasă butonul **OK**.

Interogarea Transact-SQL de ştergere a unei scheme are următoarea formă generală:

```
DROP SYNONYM [ schema. ] nume_sinonim
```

Numele schemei nu întotdeauna este un parametru obligatoriu. Dacă acesta este omis, atunci SQL Server utilizează implicit schema fixată utilizatorului curent.

Pentru a modifica un sinonim existent, se şterge cel vechi şi se creează unul nou cu proprietăţile dezirabile.

Întrebări de control:

1. Cu ce scopuri se folosesc diagramele în bazele de date SQL Server?
2 Avantajele utilizării schemelor.
3 Sintaxa de bază a instrucţiunii de creare a schemei.
4 Constrângerile posibile asupra deplasării obiectelor dintr-o schemă în alta şi asupra ştergerii schemelor.
5 Avantajele utilizării sinonimelor.
6 Sintaxa de bază a instrucţiunii de creare a sinonimelor.

Sarcini practice:

1. Creaţi o diagramă a bazei de date elaborate în capitolul 3 (sarcinile 2 şi 3), folosind forma de vizualizare standard.
2 Creaţi, în baza de date ***calculatoare*** definită în lucrările anterioare, trei scheme noi: ***stoc, pc_laptop*** şi ***copiatoare.***
3 Transferaţi tabelul ***produse*** din schema ***dbo*** în schema ***stoc,*** ţinând cont de dependenţele definite asupra tabelului ***produse***. În acelaşi mod să se trateze tabelele ***pc_uri, laptop_uri*** care aparţin schemei ***pc_laptop*** şi ***imprimante,*** care aparţine schemei ***imprimante.*** Se scrie instrucţiunile SQL respective.
4 Modificaţi interogările asupra bazei de date ***calculatoare*** prezentate în capitolul 4 astfel ca numele tabelelor accesate să fie descrise în mod explicit, ţinând cont de faptul că tabelele au fost mutate în scheme noi.
5 Creaţi sinonimele respective pentru a simplifica interogările construite în exerciţiul precedent şi reformulaţi interogările, folosind sinonimele create.

7. ADMINISTRAREA VIZIUNILOR

Capitolul tratează problematica cu privire la crearea viziunilor cu ajutorul subsistemului ***View Designer***, salvarea şi vizualizarea datelor definite în viziuni, crearea criteriilor în viziuni şi stabilirea permiselor de acces asupra viziunilor, obţinerea informaţiilor despre viziuni, precum şi crearea şi modificarea viziunilor, utilizând *Editorul de Interogări* [2].

7.1. Tabele temporare şi viziuni

Tabelele temporare sunt similare celor permanente, cu excepţia că tabelele temporare sunt păstrate în ***tempdb*** şi sunt şterse imediat când nu mai sunt folosite.

Există două tipuri de tabele temporare: *locale* şi *globale*. Ele diferă unul de altul prin numele, vizibilitatea şi disponibilitatea lor. Numele tabelelor temporare locale încep cu un simbol „#", sunt vizibile pentru utilizator numai în conexiunea curentă şi sunt şterse când are loc deconectarea de la instanţa bazei de date. Numele tabelelor temporare globale încep cu simbolurile „##". După crearea lor sunt vizibile de orice utilizator şi sunt eliminate când toţi utilizatorii ce se refereau la acestea s-au deconectat de la baza de date.

De exemplu, dacă se creează tabelul ***imprimante***, acesta poate fi folosit de orice persoană care are permisiunea în această bază de date să-l utilizeze, până la ştergerea lui din baza de date. Dacă o sesiune a bazei de date creează un tabel temporar local ***#imprimante***, numai sesiunea poate procesa acest tabel şi este eliminat când sesiunea este deconectată. Dacă, însă, se creează un tabel temporar global ***##imprimante***, fiecare utilizator al bazei de date poate procesa acest tabel. Dacă niciun utilizator nu accede la tabel după crearea acestuia, tabelul este eliminat, dacă utilizatorul ce l-a creat s-a deconectat de la baza de date. Dacă alţi utilizatori procesează tabelul global temporar, acesta va fi şters după ce ultimul utilizator se va deconecta de la baza de date.

Tabelele temporare, de obicei, se folosesc pentru păstrarea datelor provizorii şi procesarea lor ulterioara în cadrul unor proceduri mai complexe.

Mai jos, este prezentat un exemplu simplu de lucru cu tabelele temporare.

```
1  --1
2  CREATE TABLE ##imprimante_info (Tip_produs VARCHAR(30),Producator CHAR(1),Model_produs INT,
3                                  Color CHAR(2), Tip_imprimanta VARCHAR(30), Pret MONEY)
4  --2
5  INSERT INTO ##imprimante_info
6  SELECT p.Tip, p.Producator,i.Model, i.Color, i.Tip, i.Pret
7      FROM imprimante i JOIN produse p on i.model = p.model
8  --3
9  SELECT * FROM ##imprimante_info
10 --4
11 DROP TABLE ##imprimante_info
```

100 %

Results | Messages

	Tip_produs	Producator	Model_produs	Color	Tip_imprimanta	Pret
1	Imprimante	A	1276	nu	Laser	400.00
2	Imprimante	D	1433	da	Jet	270.00
3	Imprimante	E	1434	da	Jet	290.00
4	Imprimante	A	1401	nu	Matrix	150.00

Aşa cum se poate vedea din imaginea de mai sus, primul pas constă în crearea unui tabel temporar global. La al doilea pas tabelul creat este populat cu datele privind imprimantele în stoc. La al treilea pas are loc interogarea tabelului creat. Şi, în sfârşit, al patrulea pas şterge tabelul temporar creat la primul pas.

În SQL Server, o viziune (*View*) este un tabel virtual al cărui conţinut este definit de o interogare. O viziune poate fi creată din unul sau mai multe tabele ale bazei de date. Viziunile sunt frecvent utilizate

atunci când se doreşte evidenţierea unor date anumite păstrate în baza de date. Viziunile permit diferiţilor utilizatori să dispună de acces la aceleaşi date sub o optică diferită şi acestea pot fi exploatate de diverse aplicaţii.

Viziunile sunt, în general, folosite pentru a concentra, simplifica şi personaliza fiecare pătrundere a utilizatorului în baza de date. Ele pot fi folosite şi ca mecanisme de securitate, permiţând utilizatorilor să acceseze date prin intermediul viziunilor, fără a acorda utilizatorilor permisiuni de acces la tabelele din bază, în mod direct. Viziunile pot fi, de asemenea, folosite atunci când sunt copiate date din şi în SQL Server, şi pentru partajarea datelor. Există două forme de creare a viziunilor:

- Apelând la instrucţiunea *CREATE VIEW* a limbajului Transact-SQL.
- Folosind ***View Designer*** al componentei ***SQL Server Management Studio***.

Înainte de a trece la crearea viziunilor, e bine să se cunoască unele principii de creare a acestora:

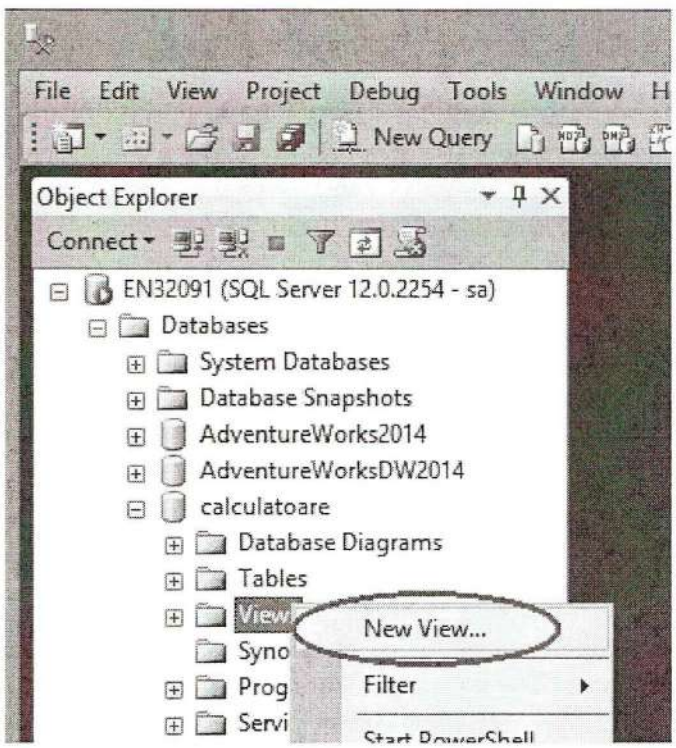
- O viziune poate fi creată numai în baza de date în uz. Totuşi, tabelele şi viziunile, ce se referă la o viziune nouă, pot exista în alte baze de date sau chiar pe alte servere.
- Pot fi construite viziuni asupra altor viziuni sau proceduri ce se referă la viziuni. Asupra viziunilor nu pot fi construiţi indecşi sau definiţi indecşi full-text.
- Nu pot fi create viziuni temporare şi nu pot fi create viziuni asupra tabelelor temporare.
- Dacă mai mult de o coloană a viziunii au acelaşi nume, ele trebuie să fie redenumite sau să fie prefixate cu numele tabelelor din care fac parte. O viziune poate avea nu mai mult de 1024 de coloane.

7.2. Administrarea viziunilor în SQL Server Management Studio

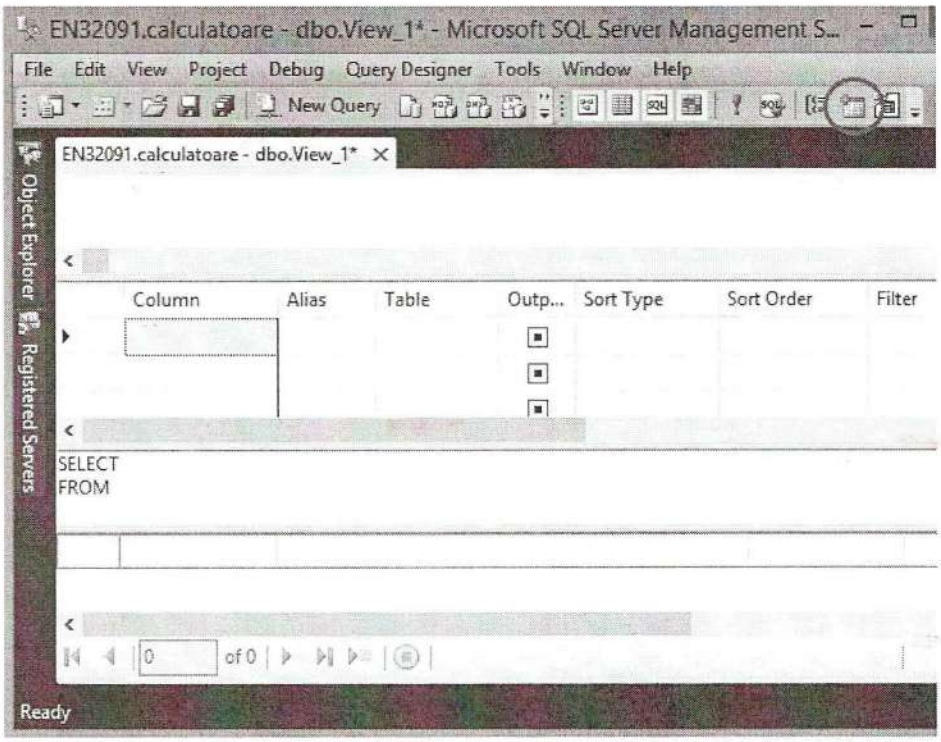
7.2.1. Crearea viziunilor cu View Designer

Pentru crearea unei viziuni, se execută următorii paşi:

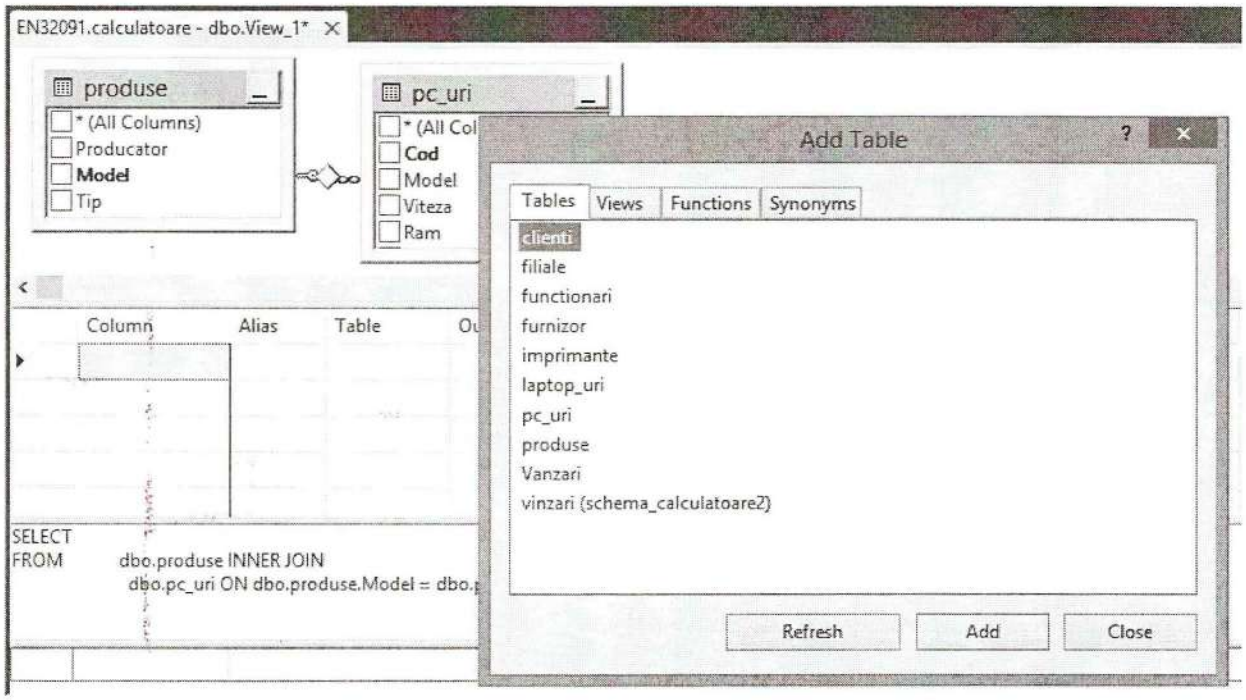
1. Se extinde grupul de servere şi apoi se extinde serverul necesar.
2. Se extinde ***Databases*** şi apoi se extinde baza de date în care se va crea viziunea.
3. Se face un click cu butonul drept al mouse-ului pe ***View***.
4. În meniul de context care a apărut, se face un click pe opţiunea ***New View...***



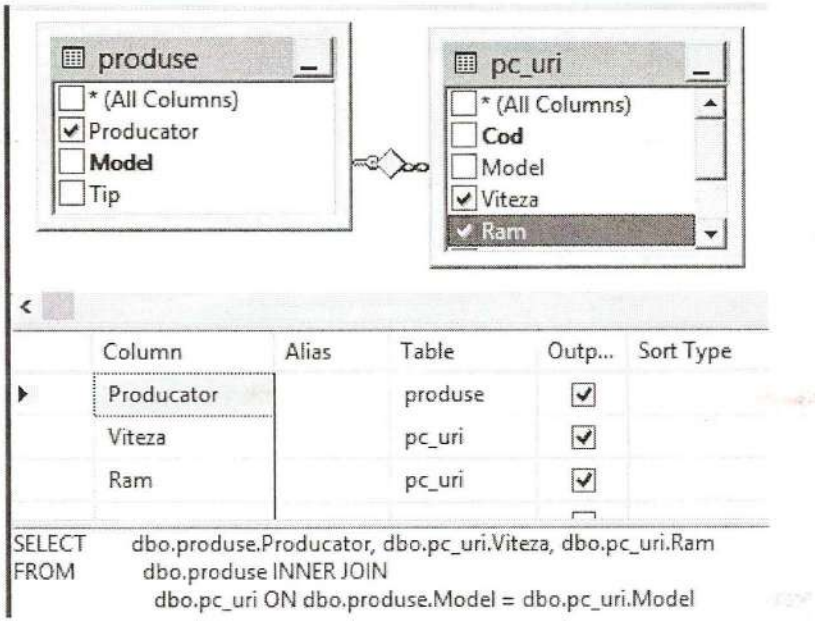
După acționarea acestei opțiuni, se deschide fereastra ce urmează. Ea este constituită din patru panele: ***Panelul diagramelor***, ***Panelul criteriilor***, ***Panelul SQL*** și ***Panelul rezultatelor***.



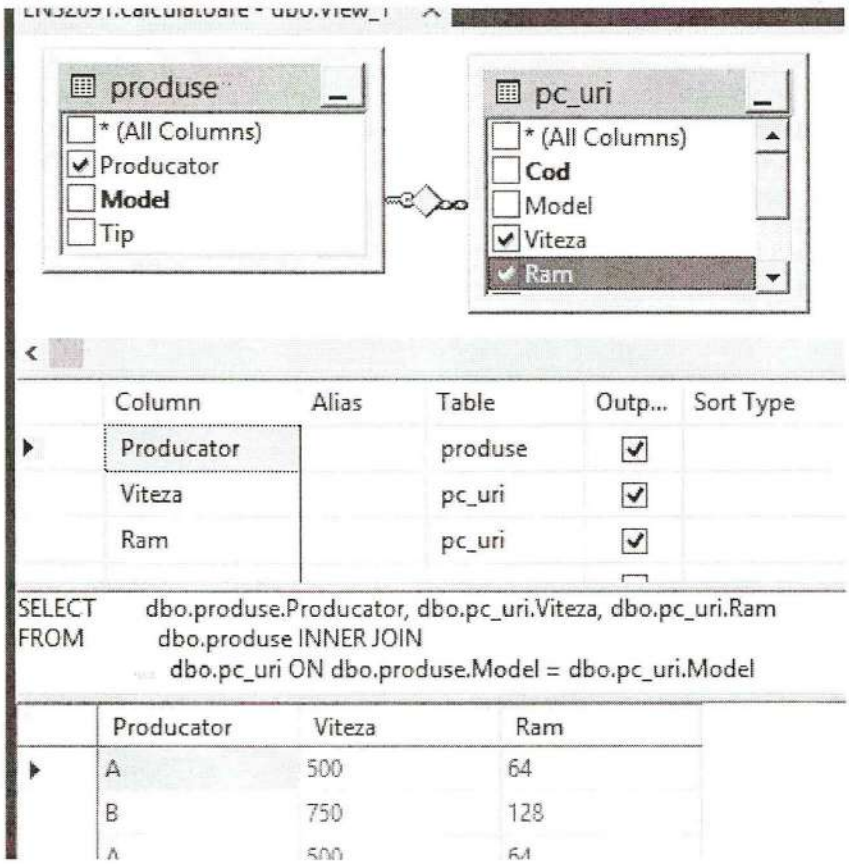
5. Se aplică un click pe butonul încercuit de pe bara de instrumente. Acesta deschide fereastra ***Add Table***. Pe filele ***Tables*** sau ***Views*** ale acestei ferestre, se dă un click pe tabelul sau viziunea care va participa la crearea viziunii și se aplică un click pe butonul **Add**. Se repetă acest proces pentru fiecare tabel sau viziune care trebuie adăugate la noua viziune. Butonul **Add** are funcția de adăugare a tabelelor care vor face parte din această viziune. Să se observe că, la adăugarea tabelelor pe panelul ***diagrame***, în panelul *SQL*, se completează automat expresia ce urmează după clauza *FROM*:



6. În panelul ***diagrame***, coloanele, care vor constitui viziunea, sunt selectate cu un click în pătrățelul corespunzător din stânga. Coloanele selectate apar automat în ***Columns*** din ***panelul criterii*** şi în panelul *SQL* urmează după clauza *SELECT*.



7. Pentru executarea viziunii, este suficientă aplicarea unui click pe butonul **Execute SQL**, de pe bara de instrumente. Rezultatul este afişat în panelul ***rezultate***:

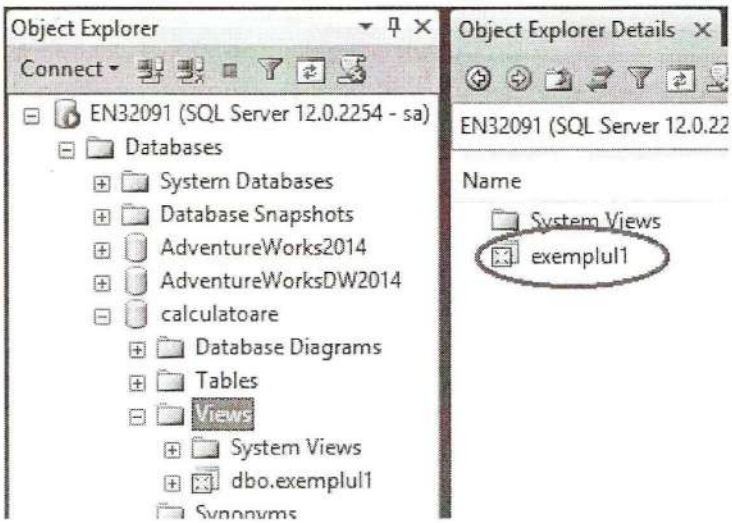


Trebuie menționat că ordinea de prezentare a coloanelor, afişată în rezultat, este determinată de secvența de clickuri de bifare aplicate asupra coloanelor din panelul ***diagrame***.

7.2.2. Salvarea viziunilor

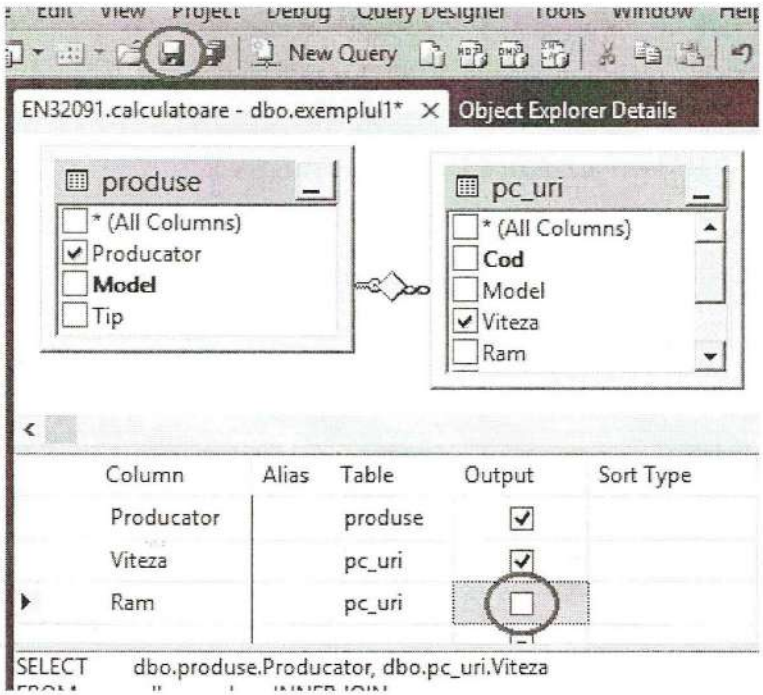
Pentru a putea fi folosită în diverse moduri, viziunea creată trebuie, mai întâi, salvată. Cu acest scop, se face un click asupra butonului **Save** şi se alege un nume pentru viziune. În cazul nostru, viziunea se va numi ***exemplul1***. Mai departe, editorul de viziuni poate fi închis.

Să se observe că s-a revenit la ***SQL Server Management Studio***, iar viziunea, deja, salvată se găseşte în mulţimea de viziuni în arborele ***Object Explorer.***



7.2.3. Excluderea unei coloane din setul de rezultate al viziunii

Pentru a exclude o coloană, se face un click cu butonul drept al mouse-ului asupra viziunii din arborele obiectelor din ***Object explorer.*** În meniul de context care s-a deschis, se aplică un click pe opţiunea ***Design***. În acest punct, va fi deschisă următoarea fereastră:



În această fereastră, este suficient să fie marcate coloanele care nu trebuie să fie prezente în rezultatul viziunii şi apoi să se salveze modificarea realizată, executând un click pe butonul **Save**. Mai departe, editorul de design al viziunii poate fi închis.

7.2.4. Modificarea ordinii de prezentare a câmpurilor

Pentru a soluționa sarcina de reordonare a câmpurilor, se aplică tehnica de glisare pe panelul *criteriilor* în fereastra deschisă anterior. După modificările efectuate, noua structură a viziunii este salvată.

7.2.5. Crearea criteriilor în viziuni

Pentru crearea criteriilor, se procedează în modul următor:

Se deschide fereastra de proiectare a viziunilor. Pe panelul criteriilor, în coloana ***Filter*** se introduce criteriul, care nu este altceva decât un filtru.

Să se execute următorul exemplu:

În coloana ***Filter*** a câmpului ***Cd***, se tipărește conținutul *LIKE '4%'* și apoi se execută viziunea, aplicând un click asupra butonului **Execute SQL**. Să se observe pe panel *rezultate*, unde sunt afișate înregistrările, că sunt expuse numai cele care au începutul valorii câmpului ***Cd*** egal cu *"4"*.

	Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...
▶	Cd		pc_uri	☑			LIKE '4%'	
	Viteza		pc_uri	☑				
	Ram		pc_uri	☑				
	Producator		produse	☑				

```
SELECT    dbo.pc_uri.Cd, dbo.pc_uri.Viteza, dbo.pc_uri.Ram, dbo.produse.Producator
FROM      dbo.produse INNER JOIN
            dbo.pc_uri ON dbo.produse.Model = dbo.pc_uri.Model
WHERE     (dbo.pc_uri.Cd LIKE '4%')
```

	Cd	Viteza	Ram	Producator
▶	40x	750	128	B
	40x	600	128	B
	40x	600	128	B
	40x	900	128	A

Acum, se formulează alt criteriu, cerând ca, în tabelul rezultant, să apară înregistrările cu valori în coloana ***Cd*** ce încep cu *'4'* și valori în coloana ***Viteza*** mai mari de 600.

	Column	Alias	Table	Output	Sort Type	Sort Order	Filter	C
	Cd		pc_uri	☑			LIKE '4%'	
▶	Viteza		pc_uri	☑			> 600	
	Ram		pc_uri	☑				
	Producator		produse	☑				

```
SELECT    dbo.pc_uri.Cd, dbo.pc_uri.Viteza, dbo.pc_uri.Ram, dbo.produse.Producator
FROM      dbo.produse INNER JOIN
            dbo.pc_uri ON dbo.produse.Model = dbo.pc_uri.Model
WHERE     (dbo.pc_uri.Cd LIKE '4%') AND (dbo.pc_uri.Viteza > 600)
```

	Cd	Viteza	Ram	Producator
▶	40x	750	128	B
	40x	900	128	A

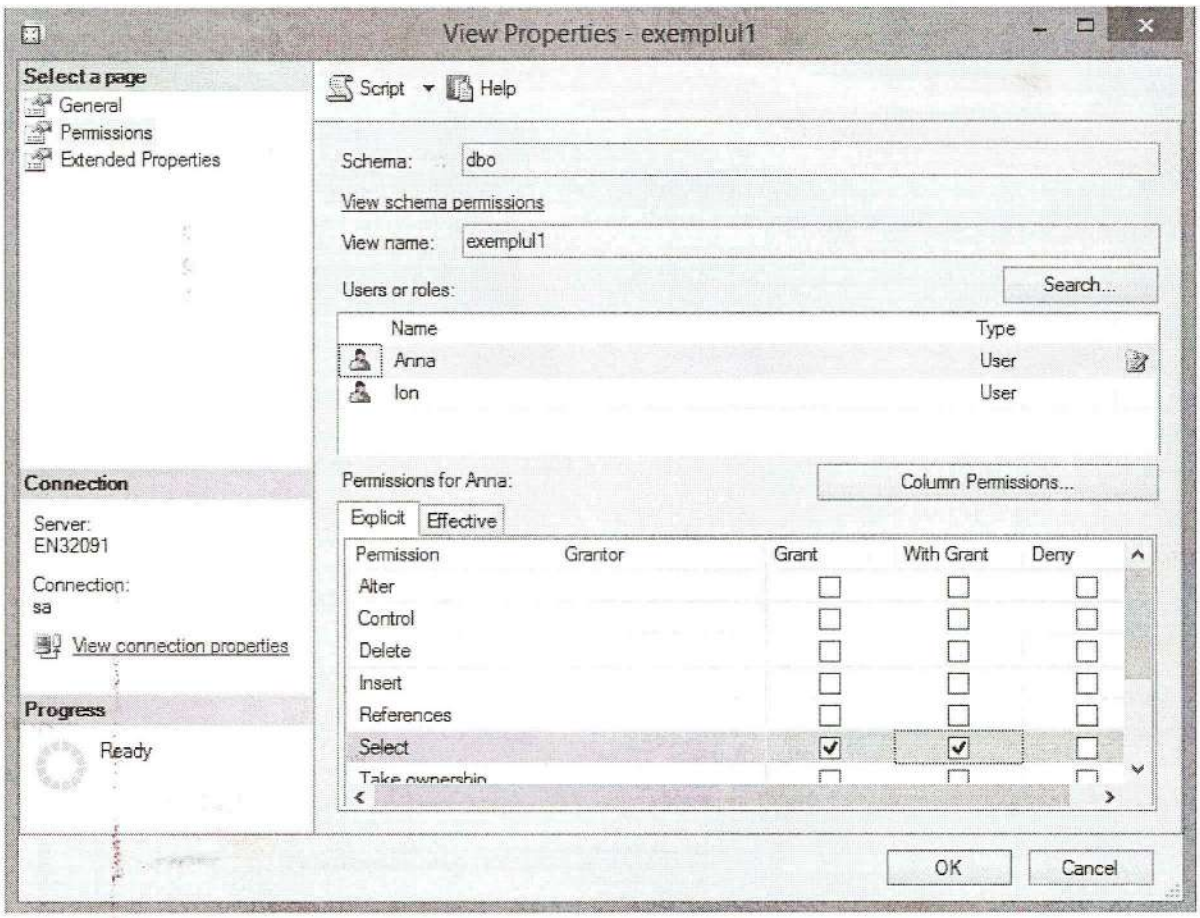
Să se observe cum este scris acest criteriu în coloana ***Filter***, în linia câmpului ***Viteza***.

Pentru a grupa după coloană sau a aplica o funcție de agregare, se dă un click cu butonul drept pe coloană și apoi în meniul de context se selectează opțiunea ***Add Group By***. După această acțiune, în panelul *criterii*, a apărut o coloană cu numele ***Group By***. În coloana ***Or***, se introduc orice criterii de specificare a liniilor căutate.

7.2.6. Acordarea permiselor

Trebuie menționat că, înainte de acordarea unui utilizator anumit a permiselor de acces la viziunile sau la alte obiecte ale bazei de date, este necesar ca acest utilizator să fie în lista de acces la serverul și/sau baza de date respective. Pentru stabilirea permiselor, se execută următorii pași:

1. În ***Object explorer*** se expandează dosarul ***Views***.
2. Se execută un click cu butonul drept al mouse-ului pe viziunea asupra căreia vor fi definite permisele.
3. În meniul de context, se face un click pe opțiunea ***Properties.***
4. Apare fereastra, deja, cunoscută, de setare a proprietăților. În partea stângă a ferestrei, se alege pagina ***Permissions*** similară celei din capitolul precedent (secțiunea *6.2.3*).
5. Pentru adăugarea accesului explicit unui utilizator sau rol, se apasă butonul ***Search***. Va apărea o fereastră de dialog ***Select Users or Roles*** și în această fereastră, pentru a deschide următoarea fereastră - ***Browse for Objects,*** se apasă butonul ***Browse***. În fereastra ***Browse for Objects,*** se alege un *rol* sau un *utilizator* care va avea acces la viziunea dată, apoi se confirmă obiectul ales prin apăsarea butonului **OK**.
6. În consecință, utilizatorii apar în tabelul ***Users or roles*** deja cunoscut. Selectând utilizatorul respectiv în tabelul ***Users or roles*** pot fi editate permisiunile de acces detaliate în tabelul de mai jos, ***Permissions for,*** fila ***Explicit.*** Pentru a viziona proprietățile native de acces al fiecărui utilizator, se alege fila ***Effective.***
7. După, se face un click pe **OK**.



7.2.7. Obținerea informațiilor despre proprietățile (dependențele) viziunilor

Se pot obține informații despre definiția unei viziuni, dacă aceasta nu este criptată. S-ar putea să fie nevoie să se vadă definiția viziunii pentru a înțelege cum sunt extrase datele ei din tabelele-sursă sau a vedea datele definite de viziune.

Când se execută o interogare printr-o viziune, motorul bazei de date execută o verifcare pentru a se asigura că toate obiectele de referință declarate există în baza de date şi că acestea sunt valabile în contextul interogării, şi că modificarea datelor respective nu încalcă nici una dintre regulile de integritate a datelor. Dacă controlul suferă eşec, el returnează un mesaj de eroare. Dacă verificarea are loc cu succes, acțiunea se transmite mai departe către tabelul sau tabelele pe care e definită viziunea.

În cazul în care o viziune este construită în baza unui tabel sau altei viziuni, care au fost şterse şi cineva încearcă să folosească această viziune, motorul bazei de date generează un mesaj de eroare. Dacă este creat un tabel nou şi structura tabelului nou nu se deosebeşte de structura tabelului precedent care a fost şters, viziunea din nou devine utilizabilă. Dacă structura tabelului nou este modificată, viziunea veche trebuie să fie ştearsă şi recreată.

În cazul în care se modifică numele unui obiect implicat în viziune, trebuie modificată viziunea astfel, încât textul ei să reflecte numele nou. De aceea, înaintea redenumirii unui obiect, mai întâi, se afişează dependențele obiectului cu scopul de a determina dacă vreo viziune este afectată de modificarea propusă.

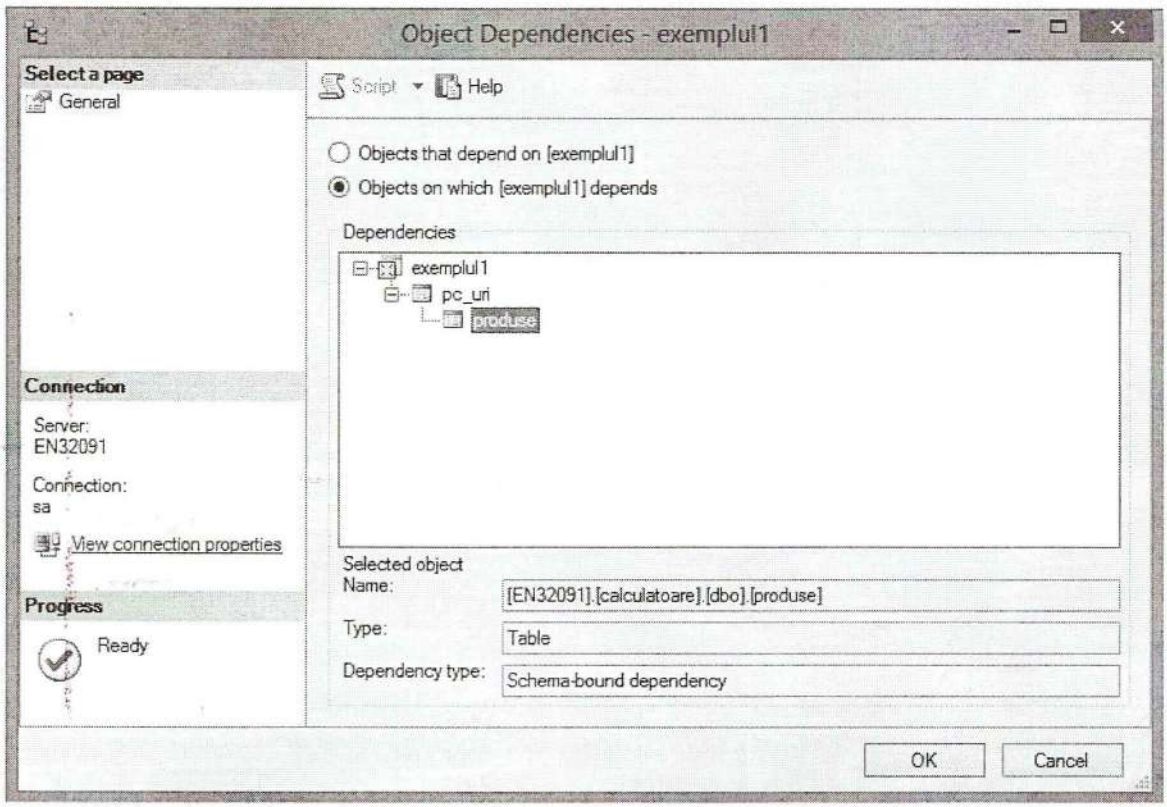
Pentru obținerea informațiilor despre viziune, se execută următorii paşi:

1. În ***Object Explorer*** se extinde un grup de servere şi apoi se extinde un server.
2. Se extinde ***Databases***, se extinde baza de date în care se găseşte viziunea şi, apoi, se extinde dosarul ***Views***.
3. Se aplică click-dreapta pe viziune şi, apoi, în meniul de context, se alege ***View Dependencies***

   În consecință, se deschide fereastra ***Object Dependencies,*** în partea de sus a acestei ferestre se află două radiobutoane:

   - ***Objects that depend on*** *<nume obiect>:* Afişează o listă a obiectelor care sunt dependente de obiectul selectat.
   - ***Objects on which*** *< nume obiect >* ***depends:*** Afişează o listă a obiectelor de care depinde obiectul selectat.

   În funcție de butonul bifat pe panelul de mai jos numit ***Dependencies,*** se afişează o ierarhie (un arbore) alcătuită din obiecte care depind de obiectul selectat sau de care acesta depinde.



Selectând un obiect sau altul din arborele-obiecte ***Dependencies***, se pot obține informații detaliate despre aceste obiecte în trei câmpuri grupate sub numele ***Selected object***.

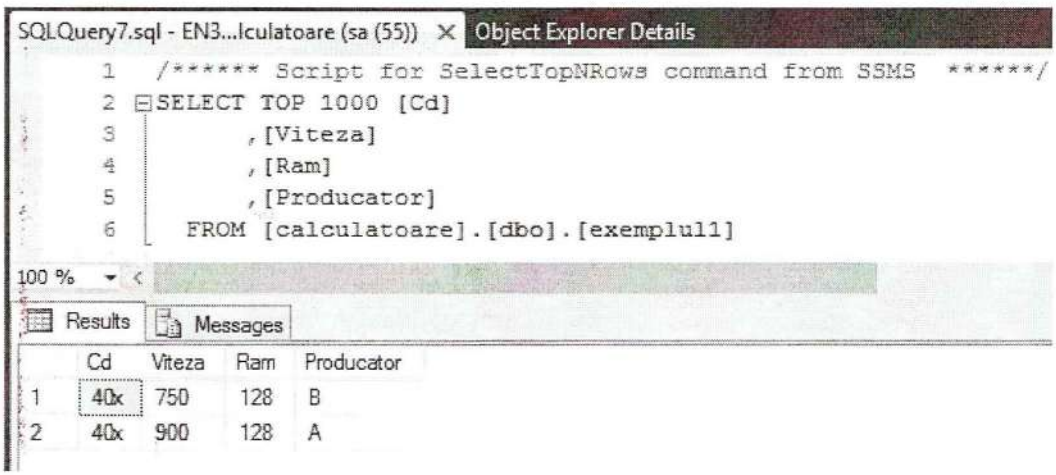
- ***Name:*** Afișează numele complet al obiectului selectat din arborele-dependențe. În cazul nostru, conform imaginii de mai sus, numele este alcătuit din *nume_server.nume_baza_de_date.nume_schema.nume_viziune*.
- ***Type:*** Afișează tipul obiectului selectat în arbore. În cazul concret, din arbore-obiecte este selectată o viziune (***exemplul1***).
- ***Dependency type:*** Afișează tipul dependenței dintre două obiecte. Tipul dependenței poate fi unul din următoarele:
  - ***Schema-bound dependency:*** Este o legătură între două obiecte care previne ștergerea sau modificarea obiectelor referite, atât timp cât există obiectul ce se referă. Dependența ***Schema-bound*** este creată atunci când o viziune sau o funcție definită de utilizator este creată, utilizând clauza *WITH SCHEMABINDING* sau atunci când un tabel referă alt obiect printr-o constrângere *CHECK* sau *DEFAULT*, sau în definiția unei coloane calculate.
  - ***Non-schema-bound dependency:*** Este o relație între două obiecte care nu împiedică obiectului referit să fie șters sau modificat.
  - ***Not available:*** Indică că tipul dependenței nu poate fi determinat. Acest lucru se produce doar atunci când este selectat un obiect de pe instanța SQL Server, versiune anterioară versiunii SQL Server 2008.

7.2.8. Vizualizarea datelor din viziuni

Se va afișa acum datele din viziuni, utilizând mediul ***SQL Server Management Studio***. Astfel, pentru consultarea unei viziuni, se execută următorii pași:

1. Se extinde grupul de servere și apoi se extinde serverul necesar.
2. Se extinde ***Databases***, se extinde baza de date în care se găsește viziunea și apoi se dă un click pe ***Views***
3. Se aplică un click cu butonul drept al mouse-ului pe viziunea în cauză.
4. Pentru a vizualiza datele definite de viziune, în meniul de context, se selectează ***Select top 1000Rows.***

   Ca rezultat, în câmpul de lucru al sistemului ***SQL Server Management Studio*** se lansează o filă nouă a ***Transact-SQL Query Editor***, similară celei prezentate în figura de mai jos. În partea de sus a filei este prezentată interogarea care cere afișarea primelor 1000 de înregistrări cu atributele ce alcătuiesc această viziune, iar în partea de jos sunt afișate rezultatele acestei interogari.



7.3. Gestiunea viziunilor, utilizând editorul de Interogări

Asupra unei viziuni, se pot efectua aceleaşi operaţii ca şi asupra unui tabel, cu deosebirea că viziunile nu conţin date şi că orice modificări efectuate asupra datelor sunt reflectate şi în viziuni. Astfel, asupra unei viziuni, se pot realiza următoarele instrucţiuni Transact-SQL: crearea viziunii (*CREATE VIEW*), modificarea viziunii (*ALTER VIEW*), ştergerea viziunii (*DROP VIEW*), interogarea viziunii (*SELECT*), adăugarea datelor (*INSERT*), modificarea datelor din viziune (*UPDATE*) şi ştergerea datelor din viziune (*DELETE*).

7.3.1. Sintaxa generală a instrucţiunii de creare a viziunilor

Sintaxa generală a instrucţiunii de creare a viziunilor este următoarea:

```
CREATE VIEW [<nume_schema>.]<nume_viziune> [ (<coloana> [ ,...n ] ) ]
    [ WITH <atribute_viziune> [ ,...n ] ]
    AS <instructiune_select>
    [ WITH CHECK OPTION ] [ ; ]

<atribute_viziunii> ::=
    {    [ ENCRYPTION ]
         [ SCHEMABINDING ]
         [ VIEW_METADATA ]    }
```

unde

`<nume_schema>`

Este numele schemei de care aparţine viziunea.

`<nume_viziune>`

Este numele asignat de utilizator viziunii care se creează. Numele viziunii trebuie să urmeze regulile de construire a identificatorilor. Specificarea numelui deţinătorului viziunii este opţională.

`<coloana>`

Este numele care va fi utilizat pentru o coloană într-o viziune. Un nume de coloană este necesar doar în cazul în care o coloană este derivată dintr-o expresie aritmetică, o funcţie, sau o constantă; atunci când două sau mai multe coloane diferite au acelaşi nume, de obicei, din cauza unei joncţiuni; sau atunci când o coloană din viziune are specificat un nume diferit de cel al coloanei din care este derivat. Nume de coloane pot fi, de asemenea, atribuite în cadrul interogării *SELECT*.

Dacă *coloana* nu este specificată, coloanele viziunii obţin aceleaşi nume ca şi coloanele respective din interogarea *SELECT*.

`<instructiune_select>`

Interogarea *SELECT* care defineşte viziunea.

`CHECK OPTION`

Forţează toate instrucţiunile de modificare a datelor care sunt executate asupra viziunii să respecte criteriile impuse de instrucţiunea *SELECT*. Atunci când o linie este modificată printr-o viziune, clauza *WITH CHECK OPTION* asigură ca datele să rămână vizibile prin viziune după ce modificarea este validată.

Orice actualizări efectuate direct asupra tabelelor care formează viziunea nu sunt verificate, chiar dacă opţiunea *CHECK* este specificată.

`ENCRYPTION`

Criptează intrările în ***sys.syscomments*** care conţin textul instrucţiunii *CREATE VIEW*. Utilizarea clauzei *WITH ENCRYPTION* previne faptul că viziunea să fie publicată ca parte a unei replicări *SQL Server*.

`SCHEMABINDING`

Leagă viziunea de schema din care fac parte tabelele în baza cărora s-a creat viziunea. În cazul în care *SCHEMABINDING* este specificat, tabelul sau tabelele de bază nu pot fi modificate într-un mod care ar afecta definiția viziunii. Însăși definiția viziunii trebuie mai întâi să fie modificată sau distrusă pentru a elimina dependențele din tabelul ce trebuie modificat. Atunci când se folosește *SCHEMABINDING*, instrucțiunea *SELECT* trebuie să includă cele două părți ale numelui (*schema.obiect*) tabelului, viziunii sau a funcției definite de utilizator care sunt referite. Toate obiectele referite trebuie sa fie în aceeași bază de date.

Viziunile sau tabelele care participă într-o viziune creată folosind clauza *SCHEMABINDING* nu pot fi distruse sau modificate înainte ca viziunea să fie distrusă sau modificată astfel, încât să nu mai existe legătura cu schema. În caz contrar, *SQL Server* va returna un mesaj de eroare.

`VIEW_METADATA`

Specifică că instanța SQL Server va returna spre *DB-Library*, *ODBC*, *OLE DB API*-uri informații de tip metadata cu privire la viziune, în loc de tabel sau tabele din baza de date, atunci când este solicitat modul de afișare de tip metadata printr-o interogare transmisă către viziune.

O viziune poate fi partiționată. O viziune partiționată reprezintă o viziune definită, folosind *UNION [ALL]* asupra tabelelor cu aceeași structură și se află pe aceeași instanță *SQL Server* sau într-un grup de instanțe *SQL Server*.

7.3.2. Crearea viziunilor

Pentru crearea unei viziuni, se utilizează instrucțiunea *CREATE VIEW*. Instrucțiunea atribuie un nume viziunii și precizează interogarea care definește viziunea. Opțional, instrucțiunea *CREATE VIEW* poate atribui un nume coloanelor viziunii nou-create. Dacă este prezentă o lista cu nume de coloane, aceasta trebuie să aibă același număr de elemente, ca și al coloanelor furnizate de interogare. Dacă lista de coloane este omisă, fiecare coloana din viziuni va avea același nume cu al coloanei din interogare. Numele coloanelor unei viziuni trebuie specificate explicit, doar în cazul în care interogarea cuprinde coloane calculate sau două coloane cu același nume, provenite din tabele-sursă diferite.

Crearea viziunii, fie ***exemplul2***, în baza de date ***calculatoare,*** poate fi realizată cu ajutorul instrucțiunii *CREATE VIEW*, de exemplu, în modul următor:

```
USE calculatoare
GO
IF OBJECT_ID ('exemplul2','V') IS NOT NULL DROP VIEW exemplul2;
GO
CREATE VIEW exemplul2 AS SELECT Model, Producator FROM produse
       WHERE Producator='A' OR Producator='B';
```

A doua instrucțiune testează existența viziunii specificate, verificând dacă posedă ID-ul respectiv. Dacă viziunea există, aceasta este suprimată. Dacă, însă, viziunea lipsește, instrucțiunea *DROP VIEW* nu este executată. *OBJECT_ID* returnează numărul de identificare al obiectului în baza de date, exprimat de schemă în calitate de argument. Parametrul „V" denotă viziune.

În a treia instrucțiune, se precizează declarația *CREATE VIEW*, după care urmează numele viziunii, ***exemplul2.*** Mai departe, e specificat cuvântul-cheie *AS*, urmat de interogarea *SELECT*, prin care se definește structura și conținutul viziunii create. Viziunea cuprinde toate înregistrările coloanelor ***Model*** și ***Producator*** din tabelul ***produse,*** furnizate de producătorii *'A'* sau *'B'*. Trebuie menționat că instrucțiunea *CREATE VIEW* nu conține, după denumirea ***exemplul2,*** coloanele necesare (***Model*** și ***Producator***), deoarece toate sunt enumerate în lista *SELECT*.

Următorul exemplu, *CREATE VIEW* ***exemplul3***, creează o viziune partiționată. Viziunea partiționată utilizează date din mai multe tabele, care au aceeași structura, din baze de date diferite sau chiar servere diferite. Un *view* partiționat poate implementa funcția *UNION [ALL]* pentru toți membri cu aceeași structură. Nu pot fi creați indecși.

Coloanele viziunii ***exemplul3*** sunt definite explicit: ***(Modele, Preturi)***. Viziunea poate fi creată utilizând una sau mai multe tabeluri şi viziuni deja definite şi asupra cărora cel care creează viziunea are drept de acces.

Opţiunea *WITH SCHEMABINDING* interzice ştergerea obiectelor, la care se referă viziunea creată. Cu alte cuvinte, structura tabelului este dependentă de cea a viziunii şi invers. Adică nu se poate modifica structura unor câmpuri care fac parte dintr-o viziune.

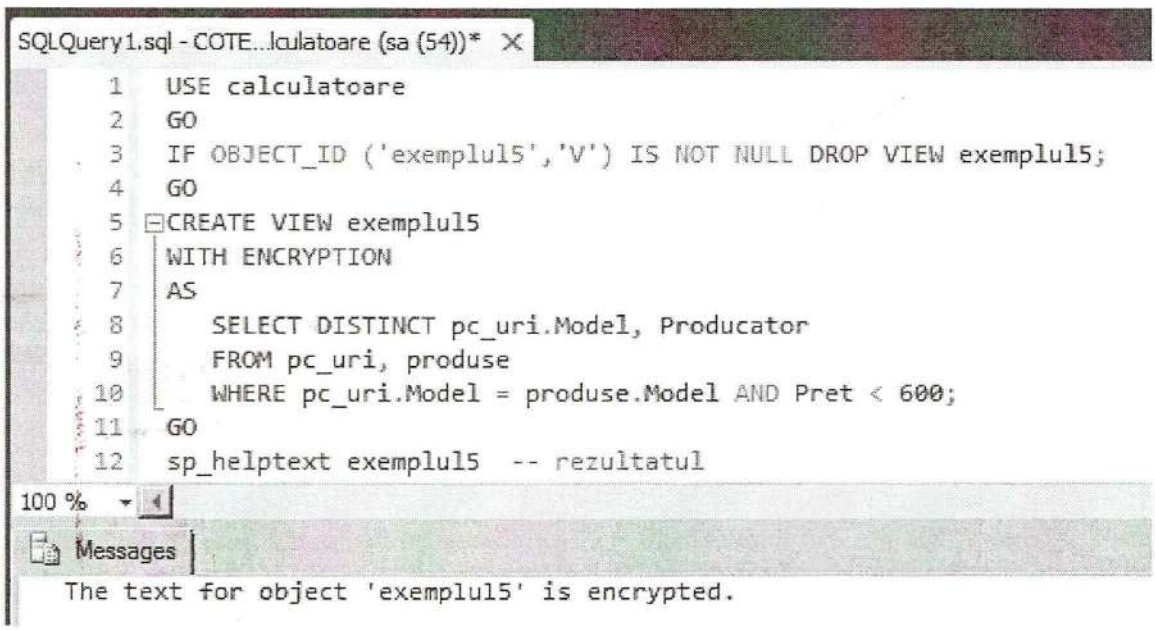
```
USE calculatoare
GO
IF OBJECT_ID ('exemplul3','V') IS NOT NULL DROP VIEW exemplul3;
GO
CREATE VIEW exemplul3 (Modele, Preturi)
     WITH SCHEMABINDING AS
     SELECT Model, Pret FROM dbo.pc_uri
          UNION
     SELECT Model, Pret FROM dbo.laptop_uri
          UNION
     SELECT Model, Pret FROM dbo.imprimante;
```

Codul următor creează o viziune, utilizând opţiunea *WITH CHECK OPTION*. Acesta obliga ca în cazul în care se folosesc comenzi de actualizare a datelor executate asupra unei viziuni trebuie să se respecte regulile de integritate impuse de instrucţiunea *SELECT*, din definirea viziunii.

În această instrucţiune, viziunea ***exemplul4*** este creată, presupunând că doar liniile care satisfac condiţia ***Tip='PC'*** pot fi inserate, actualizate sau şterse. Adică nicio instrucţiune DML nu poate să utilizeze viziunea, decât dacă criteriul din clauza *WHERE* este satisfăcut.

```
USE calculatoare
GO
IF OBJECT_ID ('exemplul4','V') IS NOT NULL DROP VIEW exemplul4;
GO
CREATE VIEW exemplul4 AS
     SELECT Producator, Tip FROM produse WHERE Tip='PC'
     WITH CHECK OPTION;
```

Următorul exemplu creează o viziune ***exemplul5*** care va conţine modelele şi producătorii de pc-uri cu un preţ sub 600 $. Definiţia viziunii nu va mai putea fi afişată:



Înregistrările tabelului ***syscomments*** conțin textul instrucțiunii *CREATE VIEW,* care poate fi criptată prin specificarea unei opțiuni. Opțiunea asigurată criptarea textului comenzii de creare a viziunii, pentru ca acesta să nu mai poată fi vizualizat și editat.

Înainte de cuvântul-cheie *AS*, este inclusă opțiunea *WITH ENCRYPTION*. Astfel, textul interogării nu poate fi redat, chiar dacă este acces deplin la baza de date și pot fi urmărite direct tabelele de sistem. Cu alte cuvinte, definiția unei viziuni protejate nu poate fi accesată nici prin intermediul administratorului de sistem SQL.

7.3.3. Operații DML asupra viziunilor

Viziunile permit programatorului Transact-SQL să creeze reprezentări ale datelor care pot fi diferite de tabelele de bază. Astfel, după ce o viziune a fost definită, ea poate fi utilizată în instrucțiunea *SELECT*, ca un tabel real. De exemplu, să se afișeze modelele produse de către producătorul *'A'*:

```
SELECT Model FROM exemplul2 WHERE Producator='A';
```

Numele viziunii, exemplul2, apare în clauza *FROM* la fel ca și numele unui tabel, iar coloanele viziunii sunt referite în instrucțiunea *SELECT* la fel ca și coloanele unui tabel obișnuit. Numai că, spre deosebire de lucrul cu tabelele, în instrucțiuni, în loc de numele tabelului, se scrie numele viziunii.

Numele unei viziuni poate fi invocat nu numai în interogări, ci și în comenzile *INSERT*, *DELETE* sau *UPDATE*, caz în care tabelele de bază din definiția view-ului pot suferi modificări și, în acest sens, se impun restricții.

Când SGBD-ul întâlnește o referință la o viziune într-o instrucțiune SQL, caută definiția acesteia în baza de date, apoi transformă cererea care face referire la viziune într-una echivalentă, dar care se referă la tabelele-sursă.

Fie asupra viziunii ***exemplul2*** era aplicată următoare instrucțiunea DML

```
UPDATE exemplul2 SET Model = '1121' WHERE Producator='C'
```

Atunci, nicio înregistrare din tabelul de bază nu mai este accesibilă pentru această viziune. Pot exista cazuri când astfel de situații sunt nedorite.

Când interogarea bazei de date se realizează printr-o viziune, sistemul verifică dacă toate obiectele referite există și dacă sunt valide în contextul instrucțiunii. Dacă cel puțin una din aceste verificări eșuează, interogarea e anulată și se returnează un mesaj de eroare.

Pentru a evita asemenea cazuri, instrucțiunea de creare a viziunii ***exemplul2*** trebuie să conțină opțiunea *WITH CHECK OPTION*. Această opțiune împiedică efectuarea de modificări ale conținutului bazei de date, dacă liniile inserate sau actualizate nu sunt regăsite prin cererea asociată viziunii.

În momentul în care în tabelele de bază sunt adăugate noi date sau sunt actualizate sau șterse cele existente, aceste modificări se reflectă corespunzător în viziunile bazate pe aceste tabele. Acest lucru este adevărat și dimpotrivă, cu singura mențiune că există anumite restricții la inserarea, modificarea sau ștergerea datelor dintr-o viziune.

Aceste restricții sunt redate pe scurt în continuare:

- Orice modificare, inclusiv prin *UPDATE*, *INSERT* și *DELETE,* trebuie să se refere la coloanele unuia și aceluiași tabel de baza.
- Coloanele modificate în viziune trebuie să se refere direct la datele din coloanele tabelelor. Coloanele nu pot fi derivate în niciun alt mod, cum ar fi prin:
  - O funcție de agregare: *AVG*, *COUNT*, *SUM*, *MIN*, *MAX*, *GROUPING*, *STDEV*, *STDEVP*, *VAR* sau *VARP*.
  - Un calcul. Coloana care nu poate fi calculată prin expresii care implică alte coloane. Coloanele formate în baza operatorilor *UNION*, *UNION ALL*, *CROSS JOIN*, *EXCEPT* sau *INTERSECT* sunt echivalente și, de asemenea, se considerate a fi calculate, și nu pot fi actualizate.

- Coloanele modificate nu sunt afectate de utilizarea clauzelor *GROUP BY*, *HAVING* sau *DISTINCT*.
- Clauza *TOP* nu este folosită nicăieri cu instrucțiunea *SELECT* în viziune împreună cu clauza *WITH CHECK OPTION*.

7.3.4. Modificarea şi suprimarea viziunilor

Instrucțiunea *ALTER VIEW* permite recompilarea unei viziuni create anterior, precum şi definirea, modificarea sau ştergerea constrângerilor asociate acelei viziuni. Atunci când se execută o instrucțiune *ALTER VIEW*, sistemul Transact-SQL recompilează viziunea, indiferent dacă este sau nu validă. Totodată, sistemul invalidează orice obiect local ataşat viziunii.

Prin recompilare se înțelege refacerea tuturor legăturilor viziunii cu tabelele de bază, verificarea validității tuturor constrângerilor şi datelor apelate şi rescrierea viziunii în sistem în caz de succes. Recompilarea unei viziuni este utilă şi pentru evitarea unor erori care pot apărea la rularea unei cereri SQL care foloseşte drept tabel viziunea.

Pentru modificarea sau recompilarea unei viziuni, se foloseşte instrucțiunea, având următoarea sintaxă:

```
ALTER VIEW [<nume_schema>.]<nume_viziune> [ (<coloana> [ ,...n ] ) ]
    [ WITH <atribute_viziune> [ ,...n ] ]
    AS <instructiune_select>
    [ WITH CHECK OPTION ] [ ; ]
```

Notațiile conțin:aceleaşi atribute ca şi în cazul instrucțiunii *CREATE VIEW*.

Se recompilează tabelul virtual ***exemplul2,*** modificând instrucțiunea *SELECT*. Viziunea este schimbată astfel, ca să conțină coloanele ***Producator***, ***Tip***, în loc de ***Model*** şi ***Producator***, şi este adăugat un filtru: ***Tip***='*PC*'.

```
ALTER VIEW exemplul2 AS
      SELECT Producator, Tip FROM produse WHERE Tip='PC'
```

Limbajul Transact-SQL oferă suport pentru suprimarea unei viziuni, prin intermediul instrucțiunii *DROP VIEW*:

```
DROP VIEW [<nume_schema>.]<nume_viziune> [ ...,n ] [ ; ]
```

Viziunile sunt şterse, aplicând numai definițiile sale, dar nu şi tabelele sau viziunile pe care se bazează. Pot fi şterse mai multe viziuni cu o singură instrucțiune *DROP VIEW*. Următorul exemplu elimină ambele obiecte:

```
DROP VIEW exemplul2, exemplul3
```

Utilizarea unei viziuni prezintă, în acelaşi timp, avantaje şi dezavantaje. Deşi viziunile constituie un instrument extrem de convenabil pentru dezvoltatorul de aplicații, ele pot avea un impact negativ asupra performanței acestora. Dacă, pentru viziuni simple (de exemplu, simple copii ale unui tabel), impactul asupra performantei nu este sesizabil, pentru viziuni complexe, care cuprind interogări pe mai multe tabele, acest impact poate fi foarte sever. De aceea, folosirea viziunilor trebuie planificată cu grijă, pentru a se vedea dacă avantajul creat de simplitatea în manipulare a acestora compensează impactul negativ asupra performanței.

7.4. Expresie-tabel

Expresia-tabel (CTE, Common Table Expression) este utilizată pentru a specifica un set temporar de rezultate. Această clauză este formată dintr-o interogare simplă şi definită în domeniul de aplicare a unei singure instrucțiuni *SELECT*, *INSERT*, *UPDATE* sau *DELETE*. Expresia-tabel, de asemenea, poate fi utilizată într-o instrucțiune *CREATE VIEW*, ca parte a instrucțiunii *SELECT* pe care o defineşte. O expresie-tabel pot include în repertoriul său referințe care se raportează la sine însăşi. În acest caz, instrucțiunea este desemnată ca o expresie-tabel recursivă.

7.4.1. Caracteristici de bază ale CTE

Expresia-tabel nu reprezintă obiecte ale bazei de date, cum ar fi viziunile sau procedurile stocate. Adresarea la o expresie CTE este posibilă numai după crearea sa.

Sintaxa generală a instrucțiunii de creare a expresiei-tabel este următoarea:

```
[ WITH <expresie_tabel> [ ,...n ] ]

<expresie_tabel>::=
    <nume_expresie> [ (<nume_coloană> [ ,...n ] ) ]
    AS
    (<definiție_interogare_CTE>)
```

`<nume_expresie>`

Identificator valid pentru o expresie-tabel. Numele argumentului `<nume_expresie>` trebuie să fie diferit de numele altor expresii-tabel definite în aceeași clauză *WITH* `<expresie_tabel>`, dar `<nume_expresie>` poate fi identică cu numele unui tabel de bază sau cu numele unei viziuni. Orice referință la argumentul `<nume_expresie>` în interogarea folosită de expresia-tabel nu este un obiect de bază.

`<nume_coloană>`

Specifică un nume de coloană în expresia-tabel. Într-o singură definiție, nume-duplicat nu sunt permise. Numărul de nume de coloane specificate trebuie să corespundă cu numărul de coloane din setul de rezultate `<definiție_interogare_CTE >`. Lista de nume de coloane nu este opțională, numai dacă numele sunt distincte pentru toate coloanele rezultate furnizate în definiția de interogare.

`<definiție_interogare_CTE >`

Specifică o instrucțiune *SELECT*, care umple setul de rezultate a expresiei-tabel. Instrucțiunea *SELECT* a `<definiție_interogare_CTE >` trebuie să îndeplinească aceleași condiții ca și cele necesare pentru a crea o viziune, cu excepția că o expresie-tabel nu poate defini o altă expresie-tabel.

În cazul în care sunt specificați mai mulți parametri `<definiție_interogare_CTE >`, definițiile de interogare trebuie să fie conectate cu unul dintre operatorii *UNION ALL*, *UNION*, *EXCEPT* sau *INTERSECT*.

7.4.2. Interogări CTE nerecursive

Rezultatele interogărilor ale expresiilor-tabel returnează o singură dată un set de rezultate pentru un apel extern. Ea nu poate conține nicio instrucțiune ce ar putea apela la propria sa expresie CTE

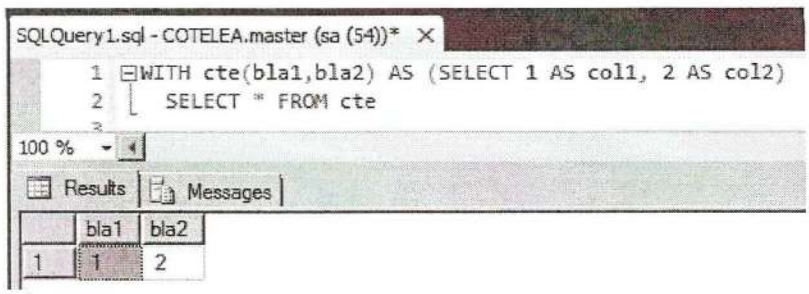
Expresiile-tabel folosesc două forme de aliasuri: externe și interne. În cazul în care coloanele sunt aliasuri în definiția de interogare propriu-zisă, atunci acesta este numit alias intern:

```
--alias coloană internă
WITH cte AS (SELECT 1 AS col1,2 AS col2)
SELECT * FROM cte
```

Dacă sunt furnizate nume de coloane în paranteze, imediat, din dreapta, după numele expresiei-tabel, atunci aceasta reprezintă aliasul extern:

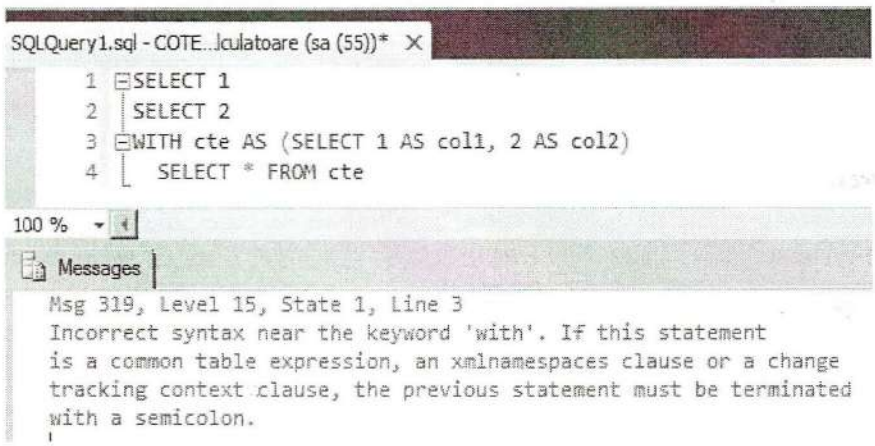
```
--alias coloană externă
WITH cte(col1,col2) AS (SELECT 1,2)
SELECT * FROM cte
```

În cazul în care sunt utilizate atât aliasurile interne, cât și externe, atunci în baza rezultatului instrucțiunii va apărea următorul tablou:



Fiecare expresie-tabel, de obicei, începe cu *WITH*. Dar dacă există cel puțin o instrucțiune înainte de *WITH*, ea trebuie să se termine cu punct și virgulă. Ce se poate întâmpla, dacă o expresie-tabel începe cu punct și virgulă urmată de *WITH* (;*WITH*)? Nu se va întâmpla nicio eroare: în acest fel, dacă se adaugă un cod înainte de expresia-tabel, ea nu va strica procesul sau scriptul SQL.

Aici, este un exemplu de expresie-tabel, în cazul în care instrucțiunea anterioară nu a fost încheiată cu punct și virgulă. Și aici apare o eroare:

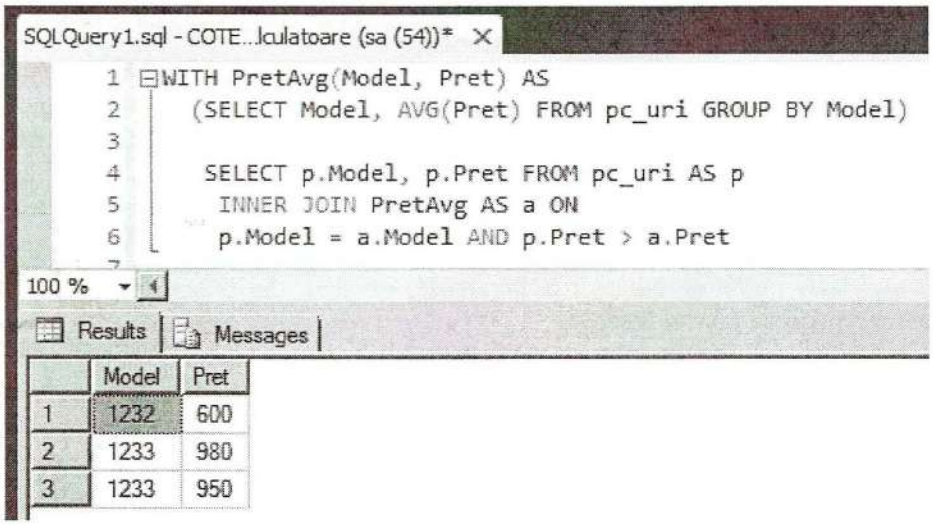


Adăugarea introducerii de punct și virgulă înainte, cu care se stabilește clauza *WITH*:

```
SELECT 1
SELECT 2
;WITH cte AS (SELECT 1 AS col1, 2 AS col2)
  SELECT * FROM cte
```

produce rezultatul scontat.

Se consideră următoarea expresie-tabel:



Linia *SELECT* calculează media aritmetică *(AVG(**Pret**))* a costurilor ***pc_urilor***, grupate după modele (***Model***).

A doua linie *SELECT* aplică joncțiunile interne, returnând liniile din tabelele asociate, care satisfac criteriile de asociere prin compararea valorilor coloanelor respective din cele două tabele. Liniile care nu satisfac criteriile sunt eliminate. Joncțiunea *INNER JOIN* returnează toate rândurile din ambele tabele, care coincid pe câmpurile de joncțiune și prețurile sunt mai mari decât prețul mediu respective.

7.4.3. Eliminarea liniilor duplicate din tabele virtuale

La proiectarea tabelelor, deseori, apare o sarcină, în special, când nu există cheia primară, dar deja datele sunt încărcate, fapt care împiedică crearea cheii. În același timp, domeniul problemei necesită unicitatea datelor.

Fie, în următorul tabel, sunt cuprinse următoarele câmpuri:

```
DECLARE @tbl_List TABLE (T_ID INT IDENTITY, T_DBMS VARCHAR(10))
INSERT INTO @tbl_List
VALUES('SQL Server'), ('Oracle'), ('PostgreSQL'), ('Oracle'),
      ('PostgreSQL'), ('SQL Server'), ('Oracle'), ('PostgreSQL');
```

Pentru simplitate, în exemplu, a fost inclusă doar o coloană, sugerând că datele sunt determinate, în mod univoc, de valorile din coloana ***T_DBMS***. Sarcina constă în construirea unei coloane ***T_DBMS*** unice (de exemplu, cheia primară), după îndepărtarea duplicatelor.

La definirea acestui tabel, a fost și introdusă coloana ***T_ID*** care se generează automat, cu alte cuvinte, ce poate specifica proprietatea de autoincrementare cu opțiunea *IDENTITY*. Această opțiune permite specificarea valorii de început a secvenței de numere crescătoare, și valoarea de incrementare care este implicit 1.

De obicei, pentru soluționarea acestei probleme, se construia un table suplimentar, după care apoi se copiau liniile unice din tabel. Dar cum poare fi soluționată această problemă fără aplicarea unui tabel auxiliar?

Se construiește o instrucțiune *SELECT* cu următoarea funcție analitică:

```
SELECT T_ID, T_DBMS,
 ROW_NUMBER() OVER(PARTITION BY T_DBMS ORDER BY T_ID) AS Num
 FROM @tbl_List;
```

Funcția *ROW_NUMBER()* atribuie un număr de ordine fiecărei înregistrări în funcție de criteriul de ordonare specificat în clauza *OVER*. Clauza *PARTITION BY* reprezintă coloana (sau coloanele) după care se creează grupări în setul de date, iar *ORDER BY* reprezintă coloana (sau coloanele) după care se realizează ordonarea pentru stabilirea valorilor-rang în cadrul partiției.

Funcția *ROW_NUMBER()* întoarce o valoare-rang dată de numărul rândului. Fiecare rând din setul de date va fi cu 1 mai mare decât precedentul rând și cu 1 mai mic decât următorul rând. Primul rând din setul de date va avea valoarea 1. Acest rang se poate aplica și pe partiții de date ale setului de date (adică, pe grupuri de date).

În cazul concret, partiția de date este realizată pe coloana ***T_DBMS***. Urmează rezultatul interogării:

	T_ID	T_DBMS	Num
1	2	Oracle	1
2	4	Oracle	2
3	7	Oracle	3
4	3	PostgreSQL	1
5	5	PostgreSQL	2
6	8	PostgreSQL	3
7	1	SQL Server	1
8	6	SQL Server	2

În exemplul de mai sus, numerotarea liniilor s-a realizat după coloana ***T_ID***. Dacă s-ar dori afișarea setului de date în altă ordine, adică adăugarea clauzei *ORDER BY*, în acest caz, numai afișarea o să se

realizeze după noua ordine, numerotarea rândurilor se va realiza după clauza *ORDER BY* a funcției *ROW_NUMBER.*

Deci, se poate proceda astfel, aplicând o expresie-tabel, care poate fi concepută și ca o reprezentare virtuală, eliminând duplicate din tabelul virtual. Deși CTE nu sunt obiecte stocate în baza de date, acestea pot utiliza operatori de actualizare. Ca urmare, totul se reduce la o singură interogare:

SQLQuery1.sql - COTE...lculatoare (sa (54))*

```
1  DECLARE @tbl_List TABLE (T_ID INT IDENTITY, T_DBMS VARCHAR(10))
2  INSERT INTO @tbl_List
3  VALUES('SQL Server'), ('Oracle'), ('PostgreSQL'), ('Oracle'), ('PostgreSQL'),
4        ('SQL Server'), ('Oracle'), ('PostgreSQL');
5  WITH CTE
6     AS
7     (
8         SELECT T_ID, T_DBMS, ROW_NUMBER()
9             OVER(PARTITION BY T_DBMS ORDER BY T_ID) AS Num
10        FROM @tbl_List
11    )
12 DELETE FROM CTE WHERE Num>1;
13 SELECT * FROM @tbl_List
```

100 %

Results | Messages

	T_ID	T_DBMS
1	1	SQL Server
2	2	Oracle
3	3	PostgreSQL

7.4.4. Interogările recursive folosind expresii-tabel

Fiind dotate cu un instrument, cum ar fi expresiile-tabel, interogările devin foarte ușor accesibile pentru rezolvarea problemei de stocare eficientă a structurilor recursive, cum ar fi lista înlănțuite, arbori, grafuri etc.

În general, se poate spune că expresiile-tabel, în comparație cu stocarea convențională a datelor recursive, ușurează operațiile de schimbare și elimină o multitudine de blocări. Dar, în același timp, sunt mai scumpe selecțiile, care necesită recursivitate: în imbricări reduce această diferență și este aproape sau complet invizibilă, dar aplicarea volumelor mari (teste empirice au arătat> 100.000), devin esențiale. În practică, aceste imbricări sunt rare, iar dacă apar vor fi necesare, probabil, alte instrumente și algoritmi.

Un avantaj al interogărilor recursive este acela că se pot auto-referi. Rostul acestora poate fi aplicat pentru a reprezenta date ierarhice așa cum este, de exemplu, o organigramă a departamentelor dintr-o companie sau un meniu pentru un site web etc.

Pentru a se rezolva acest aspect, au apărut mai mulți algoritmi. Prin interogările recursive, există posibilitatea de a standardiza modul de lucru cu datele ierarhice și cu interogări asupra acestora.

Principiul general de exprimare a recursiei:

```
WITH <nume_expresie> [ (<nume_coloană> [ ,...n ] ) ] AS
 (
   <membru_ancora>
   UNION ALL
   <membru_recursiv>
 )
SELECT * FROM <nume_expresie>
[OPTION (MAXRECURSION <numărul_bucle>)];
```

Se poate observa că interogarea recursivă este formată din două interogări numite membru ancoră şi membru recursiv. Membrul ancoră constituie interogarea al cărei rezultat va fi folosit ca intrare pentru membrul recursiv. Aceasta se va autoapela până când niciun rând nu mai este întors (condiție de încheiere). Ar mai fi menționat, ca reguli pentru interogările recursive, că *UNION ALL* este singura construcție permisă între cei doi membri şi că membrul recursiv poate referi doar o singură interogare recursivă (CTE).

În cazul în care recursia posedă o buclă infinită, se utilizează cuvântul-cheie `OPTION (MAXRECURSION <numărul_bucle>)` pentru a o evita.

Un exemplu, construit dintr-o interogare simplă cu utilizarea operatorului *UNION ALL*:

```
WITH t(N) AS (SELECT 41 UNION ALL SELECT 43) SELECT AVG(N) FROM t
```

va calcula rezultatul 42.

Pot fi declarate şi utilizate mai multe expresii-CTE. În cazul de față:

```
WITH t(N) AS (SELECT 41 UNION ALL SELECT 43),
     e(N) AS (SELECT 40 UNION ALL SELECT 44)
SELECT AVG(t.N), AVG(e.N) FROM t, e
```

rezultatul va fi o linie cu două elemente: 42, 42.

Cu toate că ambele exemple de expresii-tabel sunt legate cu operatorul `UNION ALL`, ele nu sunt recursive. O expresie-tabel recursivă trebuie cel puțin să posede un membru-ancoră şi un membru recursiv.

Expresia CTE următoare este una este recursivă:

```
WITH Litere AS
(
   SELECT ASCII('A') AS Cod, CHAR(ASCII('A')) AS Lit
   UNION ALL
   SELECT Cod+1, CHAR(Cod+1) FROM Litere
   WHERE Cod+1 <= ASCII('Z')
)
SELECT Lit FROM Litere;
```

Interogarea membrului-ancoră definește codul ***ASCII*** al primei litere a alfabetului şi simbolul ei corespunzător. Iar interogarea membrului recursiv mărește cu o unitate codul ***ASCII***, referindu-se la CTE *în propoziția FROM*. Ca urmare, în conformitate cu primul caracter va fi adăugat succesiv (*UNION ALL*) cu următoarele litere în ordinea codurilor ***ASCII***. Iterațiile vor continua atâta timp, cât condiția ***Cod*** +1 <= ***ASCII*** ('Z') este adevărată, adică, până nu se va adăuga la litera "Z".

7.4.5. O expresie-tabel recursivă pentru calculul factorialului unui număr

Un exemplu folosit pentru demonstrează puterea unei CTE este calculul factorialului, fie 5, pornind de la o singură instrucțiune Transact-SQL. În orice limbaj de programare, se face, de obicei, o buclă sau o funcție recursivă pentru a obține rezultatul.

Partea cea mai importantă este construită prin utilizarea recursiei cu expresie-tabel, aplicând *UNION ALL*. Pentru a înțelege cum ar putea arăta un program care calculează factorialul unui număr, folosind structura recursivă cu prezentarea virtuală, se aplică următorul cod:

```
SQLQuery1.sql - COTE...lculatoare (sa (54))* ×
1  WITH f(N, Fact) AS
2  (
3      SELECT 5, 1
4    UNION ALL
5      SELECT N - 1, N * Fact FROM f WHERE N > 1
6  )
7  SELECT * FROM f;
8
```

	N	Fact
1	5	1
2	4	5
3	3	20
4	2	60
5	1	120

Se analizează comparativ exemplul de mai sus cu CTE. Ordinea de executare a expresiei recursive este următoarea:

1. În CTE este membru de intrare şi se execută prima interogare:

   `SELECT 5, 1`

   Această interogare este numită membru-ancoră. Expresia-tabel se execută repetat atâta timp până când rezultatul complet este returnat. Răspunsul este stocat în rezultatul final al interogării şi în stivă.
2. Cât timp stiva nu este goală, se repetă paşii următori:
   2.1. Se execută interogarea a doua, care reprezintă membrul recursiv:

   `SELECT N - 1, N * fact FROM f WHERE N > 1`

   utilizând stiva curentă. Răspunsurile sunt introduse în rezultatul final al interogării şi în tabelul intermediar.
   2.2. Conţinutul stivei se înlocuieşte cu conţinutul intermediar, iar ultimul se curăţă

Când s-e ajunge la nivelul maxim de iterare, valorile din stivă se vor asigna rezultatului, se va combina cu membrul ancora, şi rezultatul va fi afişat. Atunci când se utilizează *UNION* (nu *UNION ALL*), la etapele 1 şi 2,1 duplicate sunt eliminate.

7.4.6. O expresie-tabel recursivă pentru parcurgerea unei ierarhii

Această procedură este un mod convenabil de parcurgere a unei ierarhii care reprezintă una dintre formele de enumerare a membrilor unei familii.

Un arbore genealogic conţine, de obicei, mult mai multe coloane decât cel prezentat aici. Este redus numărul de coloane ale tabelului pentru a înţelege construirea structurilor ierarhice virtuale. De asemenea, în această interogare, este luat în considerare doar unul dintre părinţi (tatăl sau mama) de fiecare persoană.

```
DECLARE @TT TABLE
(
 ID         INT,
 Relatie    VARCHAR(25),
 Nume       VARCHAR(25),
 ParinteID INT
)
INSERT @TT
SELECT 1, 'Străbunel',     'Gheorghe Vasilache', NULL UNION ALL
SELECT 2, 'Bunica',        'Elena Păduraru',     1 UNION ALL
SELECT 3, 'Tata',          'Ion Păduraru',       2 UNION ALL
SELECT 4, 'Unchi',         'Mihai Păduraru',     2 UNION ALL
SELECT 5, 'Tanti',         'Natalia Manole',     2 UNION ALL
SELECT 6, 'Frate bunel.', 'Mihai Vasilache',     1 UNION ALL
SELECT 7, 'Frate',         'David Păduraru',     3 UNION ALL
SELECT 8, 'Soră',          'Sorina Căruntu',     3 UNION ALL
SELECT 9, 'Frate',         'Radu Păduraru',      3 UNION ALL
SELECT 10,'Eu',            'Sandu Păduraru',     3 ;
```

Este creat şi populat un tabel în care nodul curent referă o valoare din cheia tabelului. Această structură ajută la reprezentarea unui arbore definit unic printr-o cheie, părinte şi informaţie (valoare). Se propune să se afle toate nodurile prin care se trece de la rădăcină până la frunza cu o anumită informaţie. Un exemplu mai concret ar fi acela, prin care se doreşte să se afle toţi copiii unui părinte, plecând de la părintele imediat până la străbunel sau putând afla structura ierarhică a familiei unor fraţi pornind de la acesta şi până la ultimul membru al familiei.

Pentru exemplul propus, urmează interogarea de construire a părintelui şi nivelului generaţiei pentru fiecare membru al familiei. Utilizarea expresiei-tabel recursive porneşte de la frunză cu informaţia dată.

```
WITH FamilieArb
AS
(
   SELECT *, CAST(NULL AS VARCHAR(25)) AS ParinteNume, 0 AS Generatie
   FROM @TT
   WHERE ParinteID IS NULL

 UNION ALL

   SELECT Fam.*,FamilieArb.Nume AS ParinteNume, Generatie + 1
   FROM @TT AS Fam
   INNER JOIN FamilieArb
   ON Fam.ParinteID = FamilieArb.ID
)
SELECT * FROM FamilieArb
```

Se creează membrul-ancoră, în care se formează o pseudocoloană (***Generaţie***), ce ajută să fie identificat nivelul ierarhic. Rezultatul membrului-ancoră se foloseşte ca intrare pentru membrul recursiv, se incrementează nivelul şi se afişează rezultatul:

	ID	Relatie	Nume	ParinteID	ParinteNume	Generatie
1	1	Strabunel	Gheorghe Vasilache	NULL	NULL	0
2	2	Bunica	Elena Paduraru	1	Gheorghe Vasilache	1
3	6	Frate b...	Mihai Vasilache	1	Gheorghe Vasilache	1
4	3	Tata	Ion Paduraru	2	Elena Paduraru	2
5	4	Unchi	Mihai Paduraru	2	Elena Paduraru	2
6	5	Tanti	Natalia Manole	2	Elena Paduraru	2
7	7	Frate	David Paduraru	3	Ion Paduraru	3
8	8	Sora	Sorina Carunitu	3	Ion Paduraru	3
9	9	Frate	Radu Paduraru	3	Ion Paduraru	3
10	10	Eu	Sandu Paduraru	3	Ion Paduraru	3

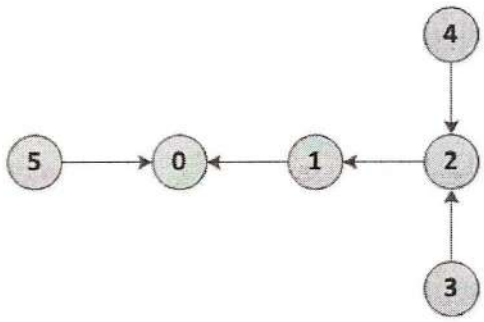
Întrebări de control:

1. Tipuri de tabele temporare. Utilitatea tabelelor temporare.
2. Viziunile şi rolul acestora în baza de date.
3. Metode şi particularităţi de creare a viziunilor în SQL Server 2014.
4. Situaţiile în care crearea viziunii este posibilă sau imposibilă.
5. Sintaxa de bază a instrucţiunii *CREATE VIEW*. Opţiunile *WITH CHECK OPTION* şi *SCHEMABINDING*.
6. Care sunt caracteristicile expresiei-tabel? Dar ale interogărilor recursive folosind reprezentări virtuale?

Sarcini practice:

1. Să se creeze două viziuni în baza interogărilor formulate în două exerciții indicate din capitolul 4. Prima viziune să fie construită în *Editorul de interogări*, iar a doua, utilizând ***View Designer***.
2. Să se scrie câte un exemplu de instrucțiuni *INSERT*, *UPDATE*, *DELETE* asupra viziunilor create. Să se adauge comentariile respective referitoare la rezultatele executării acestor instrucțiuni.
3. Sa se scrie instrucțiunile SQL care ar modifica viziunile create (în exercițiul 1) în așa fel, încât să nu fie posibilă modificarea sau ștergerea tabelelor pe care acestea sunt definite și viziunile să nu accepte operațiuni DML, dacă condițiile clauzei *WHERE* nu sunt satisfăcute.
4. Să se scrie instrucțiunile de testare a proprietăților noi definite.
5. Se consideră un graf orientat, precum cel din figura de mai jos și fie se dorește parcursă calea de la nodul id = 3 la nodul unde id = 0. Să se facă reprezentarea grafului orientat în formă de expresie-tabel recursiv.

   Să se observe instrucțiunea de după *UNION ALL* a membrului recursiv, precum și partea de până la *UNION ALL* reprezentată de membrul-ancoră.



8. PROCEDURI STOCATE ŞI FUNCŢII DEFINITE DE UTILIZATOR

Capitolul de faţă îşi propune să analizeze cele mai importante aspecte ce ţin de tipurile de proceduri stocate în SQL Server 2014, structura procedurilor stocate şi a funcţiilor definite de utilizator, crearea procedurilor stocate şi a funcţiilor definite de utilizator cu ajutorul instrucţiunilor SQL în ***Transact-SQL Query Editor***, dar şi, folosind interfaţa grafică ***SQL Server Management Studio***, precum şi executarea şi modificarea procedurilor stocate [2].

O procedură stocată sau funcţie definită de utilizator poate fi comparată cu un modul executabil al unui program. Procedura stocată sau funcţia constă într-o mulţime de instrucţiuni SQL, care realizează o anumită operaţie. Cu alte cuvinte, acestea reprezintă un program în SQL.

8.1. Noţiuni de elaborare a procedurilor stocate

8.1.1. Ce sunt procedurile stocate?

Procedurile stocate în ***Microsoft SQL Server*** sunt similare procedurilor scrise în alte limbaje de programare şi, deci, acestea:

- Acceptă parametri de intrare şi returnează multiple valori în formă de parametri de ieşire a procedurii sau grupului de proceduri.
- Conţin instrucţiuni de programare care execută operaţii asupra bazei de date, inclusiv de chemare a altor proceduri.
- Returnează valoarea de stare a procedurii sau grupului de proceduri pentru a indica succesul sau eşecul (şi cauzele eşecului) procedurii.

O procedură stocată poate fi executată manual sau poate fi invocată de alte programe. Folosirea procedurilor extinde gama de facilităţi ale mediului client-server, în principal, în problemele de executare şi menţinere a sistemului, datorită mai multor factori:

- O singură procedură stocată poate fi folosită de diverse programe.
- O singură modificare actualizează automat toate sistemele care utilizează aceeaşi procedură.
- Îndată ce procedura stocată este executată pe server, accesările ulterioare se petrec mai rapid, deoarece planul ei de executare este păstrat în memorie.

Mulţi utilizatori confundă o procedură stocată cu o funcţie, considerându-le acelaşi lucru. Procedurile stocate returnează valori prin parametrii de ieşire, şi nu o valoare în locul numelui lor, fapt ce face ca procedurile stocate să fie mai puţin utilizate în expresii.

8.1.2. Clasificarea procedurilor stocate

Procedurile stocate pot fi grupate în patru categorii principale:

- **Proceduri stocate definite de utilizator.** O procedură definită de utilizator poate fi creată într-o bază de date definită de utilizator sau în toate bazele de date de sistem, cu excepţia bazei de date ***Resource database***. Procedurile stocate pot fi elaborate în limbajul Transact-SQL sau ca o trimitere la o metodă a limbajului ***Common Runtime Language*** *(CLR)* de ***Microsoft .NET Framework***.
- **Procedurile stocate temporare** reprezintă o formă de proceduri definite de utilizator. Procedurile temporare sunt similare cu cele ale procedurilor permanente, cu excepţia faptului că sunt stocate în ***tempdb***. Există două tipuri de proceduri temporare: locale şi globale. Ele diferă unul de altul prin numele, vizibilitatea şi disponibilitatea lor.

  Procedurile temporare locale au un prefix comun unic (#). Aceste proceduri sunt vizibile numai pentru conexiunea curentă a utilizatorului şi sunt şterse de îndată ce conexiunea este închisă.

Procedurile temporare globale au prefixe formate din 2 caractere (##). Aceste proceduri sunt vizibile pentru toți utilizatorii, după ce au fost create și sunt șterse la sfârșitul ultimei sesiuni utilizând procedura.

- **Proceduri stocate ale sistemului**, care sunt create, în prealabil, și fac parte din sistemul SQL Server, și care îndeplinesc sarcini administrative. Acestea pot fi executate de orice bază de date și sunt păstrate fizic în ***Resource database*** și au prefixul ***sp_***. *Procedurile stocate ale sistemului* apar logic în ***sys schema*** fiecărui sistem de bază de date definită de utilizator. În *SQL Server 2014*, permisele *GRANT*, *DENY* și *REVOKE* pot fi aplicate în procedurile stocate ale sistemului.
- **Procedurile stocate extinse** permit crearea propriilor rutine externe în limbaje de programare cum ar fi C sau C++. *Procedurile stocate extinse* sunt *DLL-uri*, pe care o instanță poate, dinamic, să le încarce și să le execute. *Procedurile stocate extinse* se execută direct în memoria virtuală alocată pentru aplicații și sunt programate folosind ***SQL Server Extended Stored Procedure API.*** Aceste proceduri, apar pentru utilizator ca și procedurile stocate tipice și se execută în mod similar.

  Procedurile stocate extinse vor fi eliminate într-una din versiunile următoare ale sistemului SQL server. Pe viitor, Microsoft recomandă să se utilizeze procedurile CLR. Această metodă oferă o alternativă mai robustă și mai sigură de scriere procedurilor stocate.

Procedurile stocate ale sistemului sunt grupate în următoarele categorii:

Categorie	Descriere
Automation Stored Procedures	Activează folosirea obiectelor standard *Automation* într-un grup de instrucțiuni Transact-SQL.
Catalog Stored Procedures	Implementează accesul la dicționarul de date prin funcțiile ODBC și izolează aplicațiile care apelează la ODBC-uri de modificările tabelelor de bază ale sistemului
Change Data Capture Stored Procedures	Folosite pentru a activa, dezactiva, sau a raporta cu privire la modificarea datelor obiectelor capturate.
Cursor Stored Procedures	Implementează funcționalitatea cursorului
Data Collector Stored Procedures	Se utilizează pentru lucrul cu colectoarele de date și cu următoarele componente: mulțimi de colectoare, elemente ale colectoarelor și tipuri ale colectoarelor.
Database Engine Stored Procedures	Menținerea generală a motorului sistemului SQL Server.
Database Mail Stored Procedures	Utilizate pentru executarea operațiilor e-mail din cadrul unei instanțe SQL Server
Database Maintenance Plan Stored Procedures	Setează sarcinile principale de mentenanță care sunt necesare pentru gestiunea performanței sistemului.
Distributed Queries Stored Procedures	Sunt utilizate pentru implementarea și gestiunea interogărilor distribuite.
Filestream and FileTable Stored Procedures (Transact-SQL)	Utilizate pentru configurarea și gestionarea caracteristicilor *FILESTREAM* și *FileTable*
Full-Text Search Stored Procedures	Implementează și interoghează indecși *full-text*.
General Extended Procedures	Sunt utilizate cu scopul de oferire a unei interfețe la o instanță SQL Server pentru programe externe cu diverse activități de mentenanță.
Log Shipping Stored Procedures	Modifică și monitorizează mecanismul de configurare a protocoalelor tranzacționale (*log shipping*).
Policy-Based Management Stored Procedures	Se utilizează pentru administrare în baza politicilor.
Replication Procedures	Gestionează replicarea.
Security Procedures	Gestionează mijloacele de securitate.
Spatial Index Stored Procedures	Utilizate pentru analizarea și îmbunătățirea performanței indecșilor spațiali.
SQL Server Profiler Stored Procedures	Utilizată de SQL Server Profiler pentru monitorizarea performanței și activității SGBD-ului.
SQL Server Agent Procedures	Implementat de SQL Server Agent pentru gestionarea activităților planificate și orientate pe evenimente.
SQL Server Profiler Stored Procedures	Folosite de SQL Server Profiler pentru monitorizarea performanței și activității sistemului.
XML Stored Procedures	Utilizate pentru gestiunea textelor XML.

Cu excepţia unor cazuri speciale, toate procedurile stocate de sistem returnează valoarea 0. Această valoare indică faptul că procedura a fost executată cu succes. Pentru a indica eşecul, este returnată o valoare diferită de 0.

8.1.3. Reguli de creare a procedurilor stocate

Există câteva reguli de creare a procedurilor stocate.

- Definiţia de creare a procedurii poate include orice tip de numere şi orice tip de instrucţiuni SQL, cu excepţia unor instrucţiuni. Instrucţiunile, prezentate în tabelul de mai jos, nu pot fi utilizate în cadrul procedurilor stocate:

CREATE AGGREGATE	*CREATE RULE*
CREATE DEFAULT	*CREATE SCHEMA*
CREATE sau *ALTER FUNCTION*	*CREATE* sau *ALTER TRIGGER*
CREATE sau *ALTER PROCEDURE*	*CREATE sau ALTER VIEW*
SET PARSEONLY	*SET SHOWPLAN_ALL*
SET SHOWPLAN_TEXT	*SET SHOWPLAN_XML*
USE <nume baza date>	

- În cadrul unei proceduri stocate, pot fi create alte obiecte ale baze de date. Un obiect creat poate fi referit în aceeaşi procedură stocată, în cazul în care obiectul este creat înainte de referirea acestuia în procedură.
- În cadrul unei proceduri stocate, este posibilă referirea tabelelor temporare.
- Dacă se creează un tabel temporar local în interiorul procedurii stocate, el poate fi utilizat numai în cadrul procedurii date şi dispare atunci când procedura stocată este părăsită.
- Dacă se execută o procedura stocată care apelează o altă procedura stocată, atunci procedura stocată apelată poate accesa toate obiectele create de prima procedura, inclusiv tabelele temporare.
- Dacă se execută o procedură stocată îndepărtată care face modificări pe o instanţa Microsoft SQL Server îndepărtată, aceste modificări nu pot fi desfăşurate înapoi (*rolled back*). Procedurile stocate îndepărtate nu fac parte din tranzacţii.
- Numărul maximal de parametri într-o procedură stocată poate fi 2100.
- Numărul maximal de variabile locale într-o procedură stocată este limitat numai de memoria disponibilă.
- În funcţie de memoria disponibilă, dimensiunea maximă a unei proceduri stocate este de 250 de megaocteţi.

8.1.4. Structura procedurilor stocate

O procedură stocată poate fi creată, folosind instrucţiunea *CREATE PROCEDURE* şi modificată de instrucţiunea *ALTER PROCEDURE* ale limbajului Transact-SQL. Procedura stocată este compusă din nume, parametri şi corp care conţine instrucţiuni ale limbajului Transact-SQL. Instrucţiunea *CREATE PROCEDURE* nu poate fi combinată cu alte instrucţiuni ale limbajului Transact-SQL într-un singur grup.

O procedură stocată comunică cu programele apelante prin parametri. În cazul în care un program execută o procedură stocată, valorile pentru procedură sunt trecute în calitate de parametri. De exemplu, poate fi transmisă o anumită stare cu scopul de a fi folosită în instrucţiunea *SELECT*, servind, astfel, în calitate de filtru.

Un parametru trebuie să posede un nume unic, care, întotdeauna, începe cu @ şi conţine definiţia proprie a tipului de date. În continuare, este prezentat un cod al unei proceduri stocate:

```
CREATE PROCEDURE test
@Model VARCHAR(4)
```

Există, de asemenea, un mod de determinare, dacă parametrul este de intrare a datelor sau de ieşire a datelor. Pentru a-i distinge, la sfârşitul parametrului de ieşire, se adaugă clauza *OUTPUT*, ca în exemplul următor:

```
CREATE PROCEDURE test
@Model VARCHAR(4)
@Pret FLOAT OUTPUT
```

8.2. Crearea procedurilor stocate

Pentru crearea procedurilor stocate în SQL Server 2014, sunt utilizate 2 metode. O metodă presupune crearea procedurii stocate direct în ***Transact-SQL Query Editor***. O altă metodă necesită chemarea şablonului de procedură stocată din arborele de obiecte.

8.2.1. Crearea procedurilor stocate în Transact-SQL Query Editor

Se vor examina câteva exemple de creare a procedurilor stocate direct în ***Transact-SQL Query Editor***.

Mai întâi, este adusă o procedură simplă ce conţine o instrucţiune *SELECT* cu criteriu de selecţie:

```
USE calculatoare
GO
IF OBJECT_ID ('test1','P') IS NOT NULL DROP PROCEDURE test1;
GO
CREATE PROCEDURE test1
AS
SELECT Model, Pret FROM pc_uri WHERE Pret!=600;
```

A doua instrucţiune, deja cunoscută din capitolul precedent, testează existenţa procedurii stocate specificate, verificând dacă ea posedă ID-ul respectiv. Dacă procedura stocată există, aceasta este suprimată. Dacă, însă, procedura stocată nu există, instrucţiunea *DROP PROCEDURE* nu este executată. *OBJECT_ID* returnează numărul de identificare al obiectului în baza de date, exprimat de schemă în calitate de argument. Parametrul „P" denotă procedură stocată.

Această procedură stocată va afişa toate modelele de *pc*-uri şi preţurile acestora diferite de 600. Nu se utilizează niciun parametru.

În continuare, este examinată o procedură cu parametri:

```
USE calculatoare
 GO
IF OBJECT_ID ('test2','P') IS NOT NULL DROP PROCEDURE test2;
 GO
CREATE PROCEDURE test2
    @Pret FLOAT =500,
    @Model VARCHAR(4)='12%'
AS
  SELECT Model, Pret FROM dbo.pc_uri
      WHERE Pret>@Pret AND Model LIKE @Model;
```

Aici apar doi parametri ***Pret*** şi ***Viteza*** cu valori implicite *500* şi respectiv orice valoare, care începe cu simbolurile *„12"*.

A treia procedură stocată posedă parametri de ieşire.

Procedura dată afişează preţul mediu pentru fiecare producător. În funcţie de tipul produsului atribuit parametrului ***@Tip*** se execută instrucţiunea *SELECT* din tabelul respectiv. În această procedură, este creat un parametru de ieşire ***@List_mediu*** care, din start, are valoare *NULL* şi apoi se populează de instrucţiunea *SET*.

```
USE calculatoare
 GO
IF OBJECT_ID ('test3','P') IS NOT NULL DROP PROCEDURE test3;
 GO

CREATE PROCEDURE test3
      @Tip VARCHAR(10)='Laptop_uri',
      @List_mediu DECIMAL(10,2)= NULL OUTPUT
AS

IF @Tip ='Laptop_uri'
 BEGIN
   SELECT Producator,Tip,AVG(Pret)AS Pret_mediu FROM
produse,laptop_uri
    WHERE produse.Model=laptop_uri.Model AND Tip='Laptop_uri'
     GROUP BY Producator,Tip
 SET @List_mediu = (SELECT AVG(Pret)FROM laptop_uri)
 END

ELSE IF @Tip ='pc'
 BEGIN
  SELECT Producator,Tip,AVG(Pret)AS Pret_mediu FROM
produse,pc_uri
  WHERE produse.Model=pc_uri.Model AND Tip='pc' GROUP BY
Producator, Tip
 SET @List_mediu = (SELECT AVG(Pret)FROM pc_uri)
 END

ELSE IF @Tip ='imprimante'
 BEGIN
  SELECT Producator,produse.Tip,AVG(Pret)AS Pret_mediu
   FROM produse,imprimante
    WHERE produse.Model=imprimante.Model AND
produse.Tip='imprimante'
      GROUP BY Producator,produse.Tip
  SET @List_mediu = (SELECT AVG(Pret)FROM imprimante)
 END
GO
```

Precum reiese din acest exemplu, standardul Transact-SQL extinde flexibilitatea interogărilor SQL, apelând la utilizarea unor elemente din programarea procedurală.

8.2.2. Crearea procedurilor stocate din arborele de obiecte

Pentru crearea de proceduri stocate prin intermediul şablonului de procedură din arborele de obiecte, se execută următorii paşi:

1. În ***SQL Server Management Studio***, se extinde un grup de servere şi apoi se extinde un server din arborele ***Object Explorer***.
2. Se extinde ***Databases*** şi apoi se extinde baza de date în care se creează procedura.
3. Se extinde ***Programmability*** şi apoi se execută un click cu butonul drept al mouse-ului pe dosarul ***Stored Procedures***.

4. În meniul de context, care s-a deschis, se face un click asupra opţiunii ***New*** şi din elemente disponibile, se alege ***Stored Procedure...***.

   După aplicarea unui click pe opţiunea ***Stored Procedure...***, în câmpul de lucru, apare un şablon de instrucţiune de creare a unei proceduri stocate noi.

Înainte de executarea operaţiunii de creare a procedurii stocate, trebuie să ne asigurăm că procedura este corectă din punct de vedere sintactic. Pentru aceasta, e suficient să se aplice un click pe butonul **Parse**, care se găseşte pe bara de instrumente a ferestrei. După ce toate datele necesare sunt inserate în şablonul de creare a procedurii şi analiza este finalizată cu succes, se apasă butonul **Execute** de pe bara de instrumente.

8.3. Executarea procedurilor stocate

8.3.1. Executarea procedurilor stocate în Transact-SQL Query Editor

Pentru executarea procedurii stocate în ***Transact-SQL Query Editor***, se parcurg următorii paşi:

1. Se deschide Transact-SQL Query Editor.
2. În spaţiul oferit pentru tipărirea instrucţiunilor SQL, se scrie:

   EXECUTE ori *EXEC <numele procedurii stocate> <valoarea parametru>*.

În următoarea fereastră, de exemplu, se afişează instrucţiunea de executare a procedurii stocate create anterior:

SQLQuery11.sql - EN...lculatoare (sa (59))* | Object Explorer Details

```
1  USE calculatoare
2  GO
3  /* 1 */ EXECUTE test2;
4  /* 2 */ EXECUTE test2 @Pret=600;
5  /* 3 */ EXECUTE test2 600;
6  /* 4 */ EXECUTE test2 @Pret=600, @Model='1%';
7  /* 5 */ EXECUTE test2 @Model='1%', @Pret=600;
8  /* 6 */ EXECUTE test2 600, '1%';
```

100 %

Results | Messages

	Model	Pret
1	1232	600
2	1233	600
3	1233	950
4	1233	980

	Model	Pret
1	1233	950
2	1233	980

	Model	Pret
1	1233	950

În fereastră se solicită executarea procedurii stocate ***test2***. În primul exemplu, în calitate de parametri, au fost utilizate valorile implicite din procedura stocată. Exemplele 2 şi 3 sunt echivalente. În ele, pentru primul parametru, sunt definite valori diferite de cea implicită. Exemplele 4, 5 şi 6, de asemenea, reprezintă un grup de proceduri echivalente. Trebuie să se ţină cont că dacă, la chemarea procedurii stocate, nu se specifică denumirile parametrilor (ca în exemplul 6), este important să se respecte consecutivitatea valorilor în concordanţă cu consecutivitatea parametrilor.

Înainte de a trece la executarea unei proceduri stocate cu parametri de ieşire, este important să se menţioneze că parametrii de ieşire permit procedurii stocate să returneze datele către aplicaţia-client care a iniţiat executarea acestei proceduri. Aici, indicarea cuvântului-cheie *OUTPUT* este obligatorie la crearea procedurii şi la apelul acesteia. Parametrii de ieşire în cadrul unei proceduri stocare sunt considerate drept variabile locale. Pentru a obţine o valoare (rezultat) în procedura care apelează, variabilele de ieşire mai întâi trebuie definite. Când executarea procedurii stocate ia sfârşit, valoarea curentă a parametrului de ieşire este transmisă variabilei locale din programul care a iniţiat executarea procedurii stocate.

Urmează un exemplu cu utilizarea parametrilor *OUTPUT*:

SQLQuery11.sql - EN...lculatoare (sa (59))* | Object Explorer Details

```
1  USE calculatoare;
2  GO
3  DECLARE @List_mediu FLOAT
4  EXECUTE test3 'pc',@List_mediu OUTPUT
5    IF @List_mediu>250
6      BEGIN
7      PRINT 'Pretul mediu pe tabelul respectiv este egal cu '
8                    +RTRIM(CAST(@List_mediu AS varchar(10)))+'$ .'
9      END;
10 GO
```

100 %

Results

```
Producator Tip        Pret_mediu
---------- ---------- ----------------------
A          PC         604.285714285714
B          PC         850
E          PC         350

(3 row(s) affected)
Pretul mediu pe tabelul respectiv este egal cu 648.18$ .
```

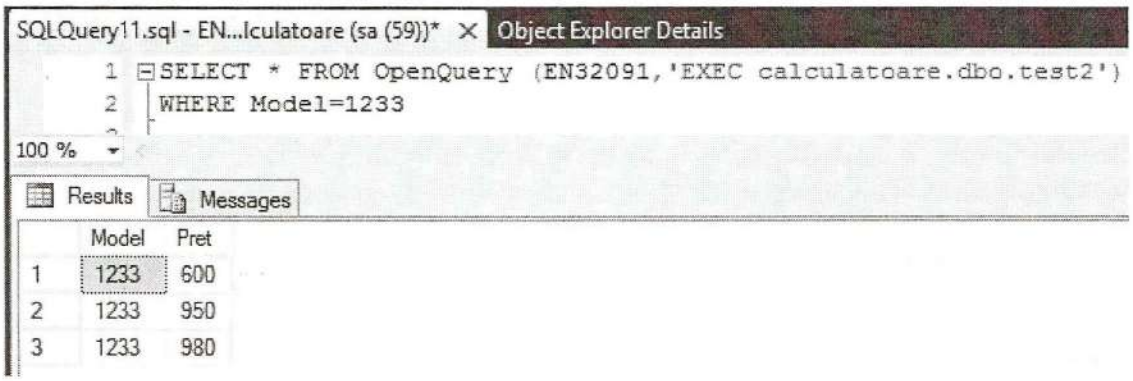
În exemplul dat, în calitate de parametru de ieşire este definit ***@List_mediu***. La chemarea procedurii pentru parametrul de intrare ***@Tip*** era definită valoarea egală cu *'pc'*. Mai departe, este specificat că, dacă valoarea luată de parametrul ***@List_mediu*** va depăşi 250, se va afişa un mesaj cu valoarea curentă a acestui atribut. Pentru a putea fi inclusă în mesajul textual, valoarea variabilei ***@List_mediu*** este convertită de către funcţia *CAST* la tipul de date *VARCHAR*.

Precum se poate observa din exemplele anterioare, procedurile stocate sunt chemate spre executare prin intermediul comenzii *EXEC* dintr-un pachet de instrucţiuni Transact-SQL sau dintr-o aplicaţie-client. În acelaşi timp o procedură stocată poate fi chemată şi dintr-o instrucţiune, după clauza *FROM*, utilizând funcţia ***OpenQuery()***.

De regulă, funcţia ***OpenQuery()*** este utilizată pentru executarea instrucţiunilor SQL pe un server îndepărtat. Dacă în funcţia ***OpenQuery()*** este specificat numele procedurii stocate, atunci rezultatele executării acestei proceduri sunt transmise către serverul local.

Deoarece setul de date obţinut în calitate de rezultat al procedurii stocate este utilizat de clauza *FROM* din instrucţiune *SELECT*, predicatul ce urmează după clauza *WHERE* poate reduce volumul de date de ieşire oferit de procedura stocată.

În exemplul ce urmează se execută o procedură stocată pe un server specificat de parametrii funcţiei. Asupra rezultatului returnat de procedura stocată este aplicat un filtru suplimentar din clauza *WHERE*. Funcţia ***OpenQuery()*** are doi parametri: numele serverului conectat şi instrucţiunea care va fi executată pe serverul dat.



Cu toate că această tehnică permite utilizarea procedurilor stocate în instrucţiunile *SELECT*, nu este atât de optimizată ca tehnica de transmitere a constrângerilor pentru prelucrare de către procedura stocată. Există doar un singur avantaj evident - posibilitatea de chemare a procedurilor stocate de orice complexitate din orice instrucţiune *SELECT*.

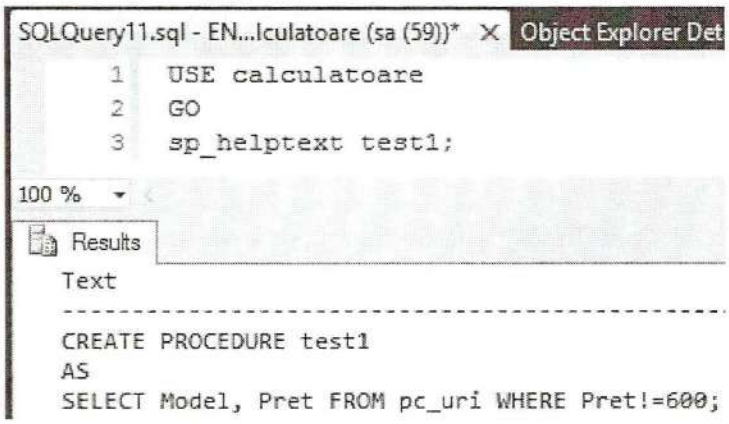
Pentru executarea cu succes a funcţiei ***OpenQuery()***, este important ca pe serverul la care se conectează sistemul pentru executarea funcţiei să fie configurate corect parametrii *DATA ACCESS*.

Datele returnate de o procedură stocată pot fi păstrate pentru procesarea ulterioară aşa cum este demonstrat în exemplul de mai jos. La început, este creat un tabel temporar care, ulterior, este populat cu datele returnate de procedura stocată. Apoi datele din tabelul dat pot fi utilizate ori de câte ori este nevoie în cadrul sesiunii curente.

```
USE calculatoare
GO
CREATE TABLE #sp_output (Model INT, Pret MONEY);
INSERT INTO #sp_output EXEC test2;

SELECT * FROM #sp_output;
```

În cazul în care se creează o procedură stocată, textul ei este salvat în tabelul ***SysComments***. Acest text nu se salvează cu scopul executării procedurii, ci cu scopul modificării posibile ulterioare. Procedura stocată de sistem ***sp_helptext*** poate extrage codul iniţial al procedurii stocate din tabelul ***SysComments***, precum este prezentat în figura de mai jos:



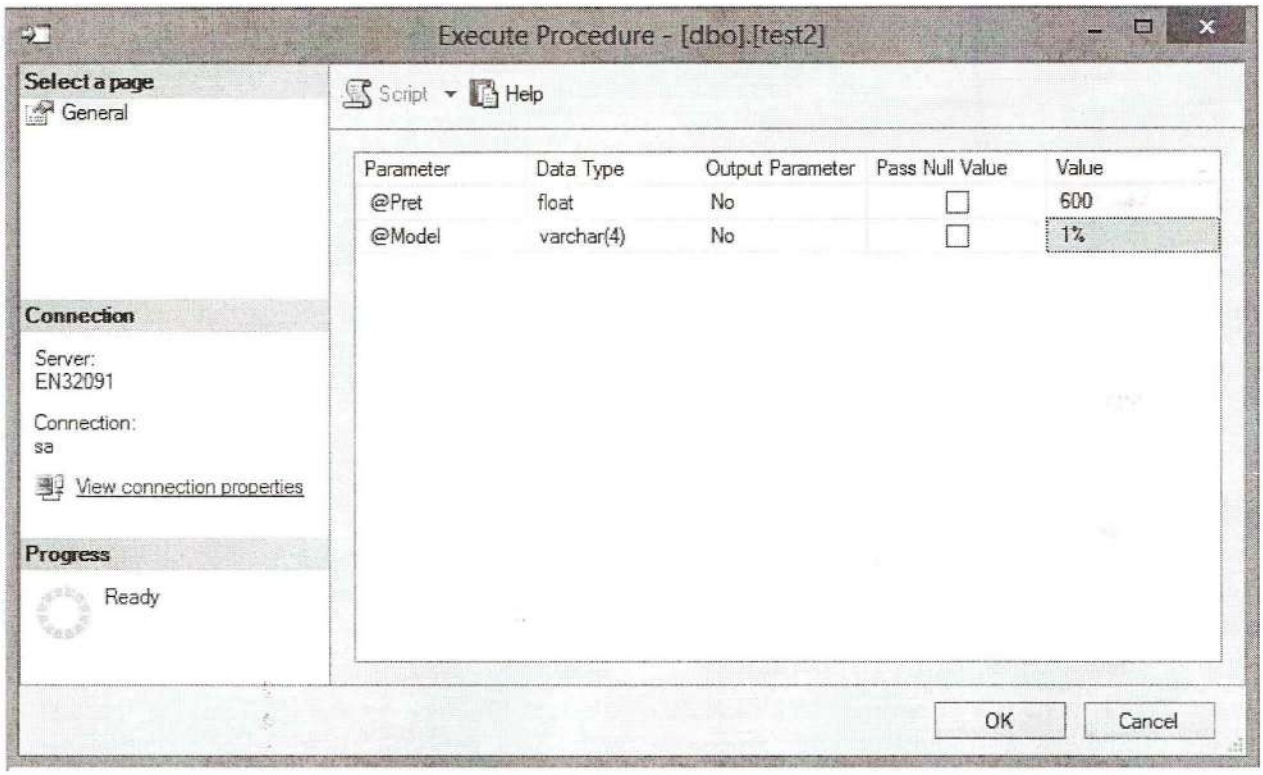
8.3.2. Executarea procedurilor stocate în SQL Server Management Studio

Pentru executarea procedurii stocate, folosind interfaţa ***SQL Server Management Studio***, se parcurg următorii paşi:

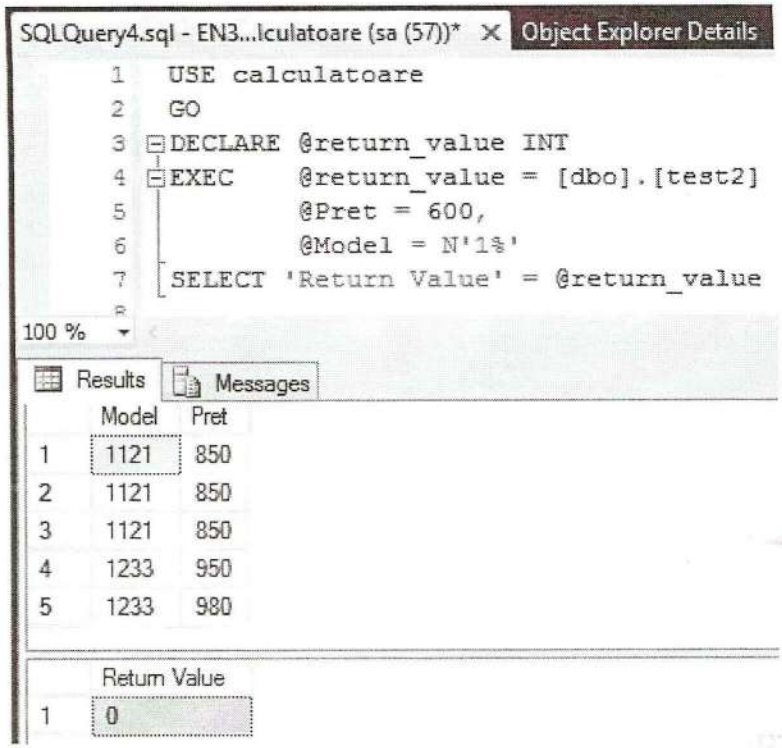
1. În ***SQL Server Management Studio***, se extinde un grup de servere şi apoi se extinde un server din ***Object Explorer***.
2. Se extinde ***Databases*** şi apoi se extinde baza de date în care se creează procedura.

3. Se extinde ***Programmability*** şi apoi se execută un click pe dosarul ***Stored Procedures***.
4. Apoi, în lista de proceduri stocate ce apare în câmpul de lucru, se face un click-drept pe procedura necesară
5. În meniul de context apărut, se alege ***Execute stored procedure***

În consecinţă, apare următoarea fereastră în care pot fi setate valorile parametrilor de intrare (coloana ***Value***). Sau sunt lăsate vide. În acest caz, sistemul va utiliza valorile implicite, dacă ele sunt definite în procedura stocată respectivă. Dacă nu, atunci, pentru a evita erorile de executare, completarea acestui câmp devine obligatorie.



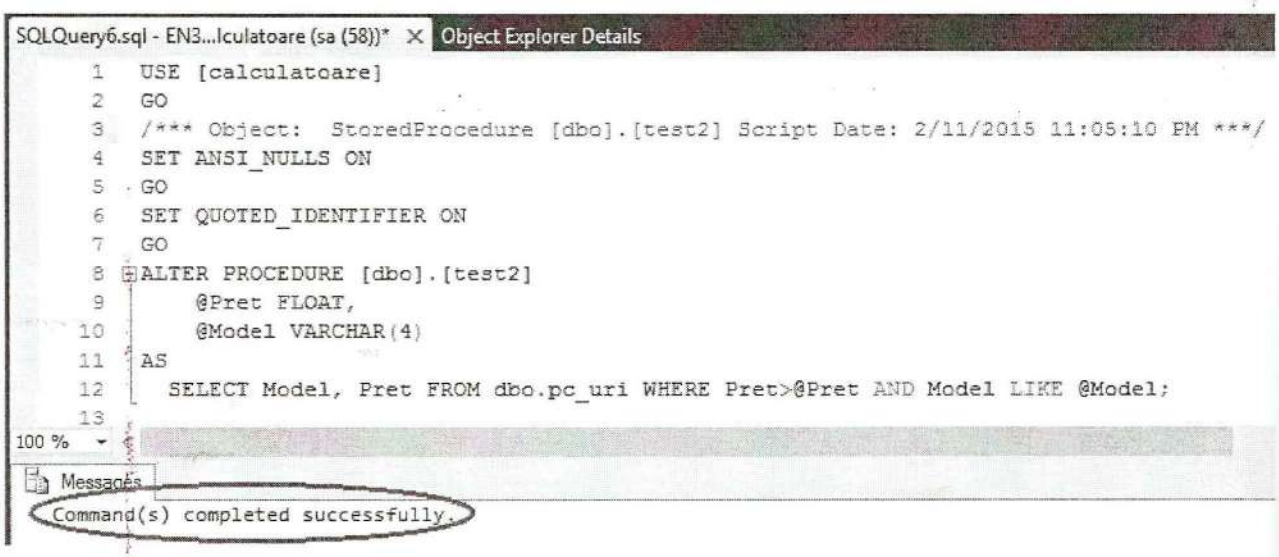
Dacă se bifează în coloana ***Pass Null Value,*** sistemului i se comunică faptul că pentru parametrii respectivi sunt setate valori nule. În câmpul ***Value*** se setează valorile pentru parametri de intrare diferite de cele implicite. După completările necesare, se apasă butonul **OK**. Rezultatul executării se afişează în mediul ***Transact-SQL Query Editor***, precum se poate observa în exemplul de mai jos:



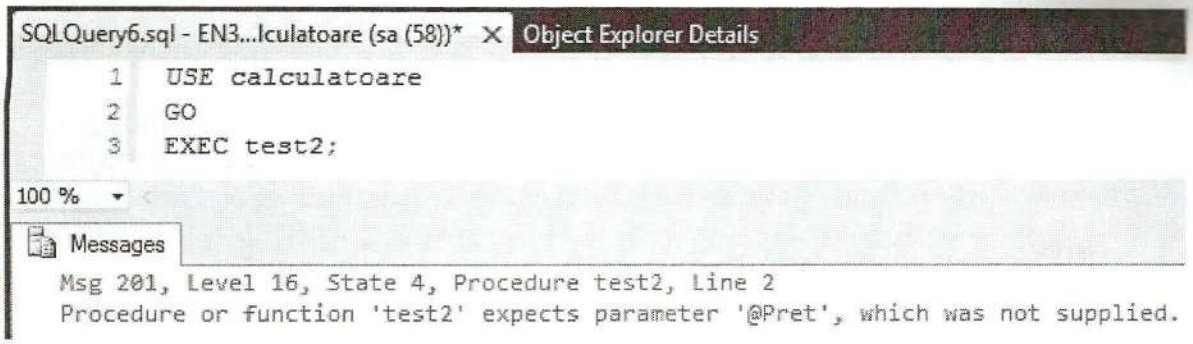
8.4. Modificarea procedurilor stocate

Modificarea procedurii stocate, folosind interfaţa ***SQL Server Management Studio***, este similară cu executarea procedurii stocate numai că, în locul selectării opţiunii ***Execute stored procedure*** în meniul de context apărut, se alege ***Modify***.

În cazul în care se aplică un click pe opţiunea ***Modify***, automat se deschide mediul ***Transact-SQL Query Editor***, unde pot fi realizate modificările considerate necesare. În exemplul dat, se exclude o valoare prestabilită a parametrilor din procedura ***test2***. Aici, definirea parametrilor de intrare este obligatorie şi în cazul când utilizatorul uită să-i introducă, sistemul va detecta o eroare. După modificările necesare, se apasă butonul ***Execut.*** În consecinţă, pe fila ***Results*** trebuie să apară un mesaj de executare reuşită.



Să se observe cum arăta instrucțiunea după excluderea valorilor implicite. Acum se lansează spre executare procedura respectivă:



Executarea procedurii a generat un mesaj de eroare, deoarece nu erau specificate valorile parametrilor de intrare.

8.5. Funcții definite de utilizator

În afara de setul de funcții predefinite, SQL server permite utilizatorilor să definească propriile funcții prin instrucțiuni Transact-SQL.

O funcție SQL, definită de utilizator, grupează în cadrul unui obiect SQL de sine stătător o secvență de instrucțiuni SQL încapsulată, ce poate fi reutilizată ori de câte ori este nevoie, în cadrul altor obiecte SQL Server.

Funcțiile definite de utilizatori au rolul de a returna un rezultat, calculat conform unui algoritm descris de utilizator și pot utiliza parametri în cadrul calculelor.

În funcție de tipul de rezultat returnat, funcțiile definite de utilizatori pot fi clasificate în două mari categorii: funcții scalare, funcții de tip tabel simplu (*Inline table-valued*) și funcții de tip tabel multi-instrucțiune (*Multi-statement table-valued*).

8.5.1. Funcții de tip scalar

Funcțiile scalare returnează un singur rezultat scalar. Acest rezultat poate fi de orice tip, cu excepția tipurilor *TEXT*, *NTEXT*, *IMAGE* și *TIMESTAMP*. Ea reprezintă cea mai simplă formă a funcției.

Pentru a crea o astfel de funcție, se va utiliza instrucțiunea *CREATE FUNCTION*, a cărei sintaxă simplificată este următoarea:

```
CREATE FUNCTION [<nume_schemă>.]<nume_funcție>
( [ { <@nume_parametru> [AS] [<tip_nume_schema>.]<tip_date_parametru>
          [=<valoare_implicita>] [ READONLY ]} [ ,...n ] ] )
RETURNS <tip_date_returnat>
   [WITH <opțiune_funcție> [ ,...n ] ]
   [AS]
   BEGIN
     <corp_funcție>
     RETURN <expresie_scalar>
   END
```

unde:

- `<nume_funcție>` - declararea numelui funcției.
- `<nume_schemă>` - numele schemei existente, care posedă funcția definită de utilizator.
- `<@nume_parametru>` - reprezintă numele unui parametru formal din lista de parametri.
- `[<tip_nume_schema>.]<tip_date_parametru>` - reprezintă tipul unui parametru formal și, opțional, numele schemei lui.

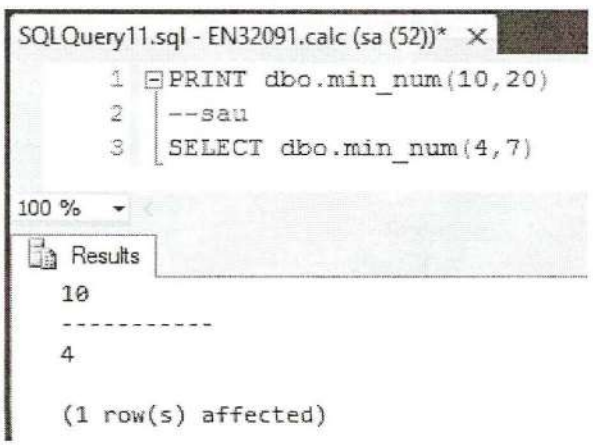
- =<valoare_implicita> - exprimă o valoare implicită pentru parametru. Funcţia poate fi executată fără a fi specificată o valoare pentru acest parametru.
- READONLY - indică faptul că parametrul nu poate fi actualizat sau modificat în definiţia funcţiei. În cazul în care parametru este de tip tabel definit de utilizator, trebuie să fie precizat cuvântul-cheie *READONLY*.
- WITH <opţiune_funcţie> - pentru a specifica diverse opţiuni pentru funcţie. De exemplu, opţiunea *ENCRYPTION* criptează codul din tabelul de sistem, iar *SCHEMABINDING* permite să lege funcţia cu toate obiectele bazei de date (schema) la care se referă. În acest caz, suprimarea şi actualizarea oricărui obiect al bazei de date, legătura cu funcţia este imposibilă fără schimbarea propriei funcţii.
- <tip_date_returnat> - se precizează tipul de date al valorii returnate de funcţie.
- <expresie_scalar> - valoare de returnat.
- <corp_funcţie> - în funcţii scalare, reprezintă un set de instrucţiuni Transact-SQL, care, împreună, returnează o valoare scalară.

Mai întâi, se consideră următorul exemplu - o funcţie scalară. Se creează o funcţie, care, din două numere întregi, aplicate în calitate de parametri de intrare, va alege cel mai mic:

```
USE calculatoare
GO
IF OBJECT_ID ('min_num','FN') IS NOT NULL DROP FUNCTION min_num
GO
CREATE FUNCTION min_num(@a INT, @b INT)
RETURNS INT
     BEGIN
           DECLARE @c INT
           IF @a<@b SET @c=@a
            ELSE SET @c= @b
           RETURN @c
     END
```

După clauzele *CREATE FUNCTION*, se specifică numele funcţiei. Mai departe, urmează parametrii, care se dau între paranteze rotunde, toate despărţite prin virgulă. Această prezentare este diferită de cea a procedurilor stocate. Apoi, după cuvântul-cheie *RETURNS* se precizează tipul de date de tip scalar al valorii returnate de funcţie. În corpul funcţiei, între *BEGIN* şi *END* este inclusă structura alternativă *IF...ELSE* şi declaraţii de variabilă, *@c*.

Odată ce funcţia este creată, poată fi folosită pentru a returna valoarea:

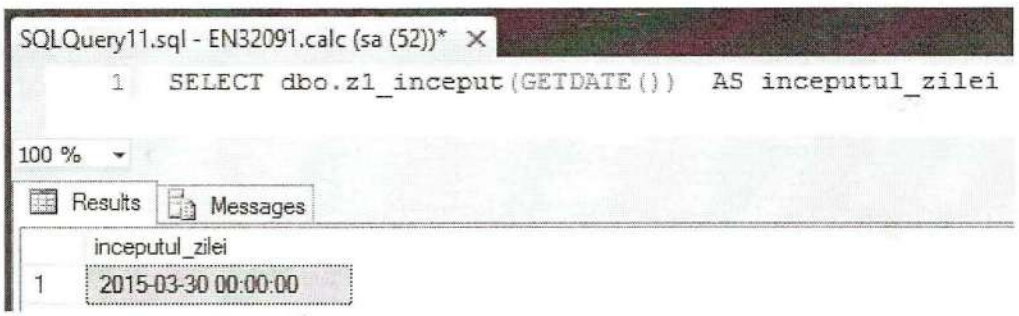


Se doreşte să se realizeze o funcţie care va recepţiona un parametru de intrare de tip *DATETIME* şi va returna ora de la începutul zilei. De exemplu, în cazul în care parametrul de intrare este 25.03.15 13:31 (13:31 25 martie 2015), rezultatul va fi începutul din 25 martie - 25.03.15 0:0.

```
IF OBJECT_ID ('z1_inceput','FN') IS NOT NULL DROP FUNCTION z1_inceput
GO
CREATE FUNCTION dbo.z1_inceput (@dat DATETIME)
RETURNS SMALLDATETIME AS
BEGIN
    RETURN CONVERT(DATETIME, FLOOR(CONVERT(FLOAT, @dat)))
END
```

Funcţia *CONVERT* converteşte un număr dintr-un sistem de măsură în altul (de exemplu, *CONVERT* poate transforma un tabel cu distanţe în mile într-un tabel cu distanţe în kilometri). Mai întâi, tipul de date *DATETIME* este transformat în tip *FLOAT*. Cu această formă transformată, partea întreagă reprezintă numărul de zile, începând cu 1 ianuarie 1900, iar partea fracţionară - timpul. Mai departe, se petrece o rotunjire până la cel mai mic număr întreg, folosind funcţia *FLOOR* şi, din nou, se aduce la tipul de format *DATETIME*, dar timpul deja a căzut la 0:0.

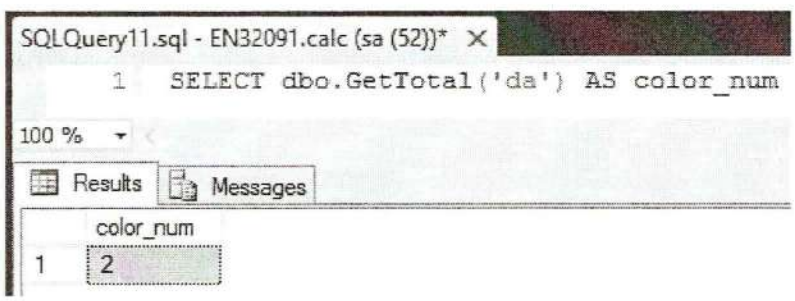
Urmează un exemplu de testare a funcţiei, unde funcţia *GETDATE*() – întoarce data şi ora curentă a sistemului:



Funcţiile anterioare utilizau în calcul numai parametri de intrare. Dar pot fi folosite şi datele stocate în baza de date. Fie, se doreşte să se realizeze o funcţie, care va avea un singur parametru – care specifică dacă imprimanta este în culori sau nu – şi va calcula numărul total de imprimante.

```
IF OBJECT_ID ('GetTotal','FN') IS NOT NULL DROP FUNCTION GetTotal
GO
CREATE FUNCTION GetTotal(@Color CHAR(2))
RETURNS INT
 BEGIN
  DECLARE @Total INT
  SELECT @Total = COUNT(*)
      FROM imprimante
      WHERE Color = @Color
  RETURN @Total
 END
```

În acest exemplu, *SELECT* returnează rezultatul executării funcţiei *GetTotal*. Funcţia aparţine schemei **dbo**. După nume, în paranteze sunt enumeraţi parametrii în aceeaşi ordine ca şi în declaraţia funcţiei.



Numele complet al unei funcţii este dat de următorul specificator:

```
NumeBazaDate.NumeSchemăFunctie.NumeFuncţie
```

Trebuie menţionat faptul că o funcţie de tip scalar poate accepta zero sau mai mulţi parametri şi va returna o singură valoare. Sunt folosite, de regulă, pentru a converti o valoare într-alta. Funcţia scalară nu poate fi folosită în clauza *FROM* a unei instrucţiuni *SELECT*, însă pot fi utilizate în expresie-coloană în *SELECT* sau *GROUP BY*, condiţiile *WHERE* sau *HAVING*.

8.5.2. Funcţii de tip tabel simplu

Funcţia de tip Inline nu returnează în calitate de rezultat o valoare scalară, ci întregul tabel, sau, mai degrabă, un set de date. Acest lucru poate fi foarte convenabil în cazul în care în diverse proceduri, declanşatoare etc. se aplică una şi aceeaşi subinterogare. Astfel, în loc de a scrie această interogare, poate fi creată o funcţie şi folosită în continuare. Funcţia devine şi mai utilă, în cazul în care tabelul returnat este dependent de parametri de intrare. După cum se cunoaşte, viziunile nu pot avea parametri, prin urmare, această problemă poate fi soluţionată cu ajutorul funcţiei de tip tabel. Caracteristica principală a funcţiei de tip tabel simplu constă în faptul că funcţia poate conţine numai o singură interogare în corpul său. Adică, funcţiile Inline sunt foarte asemănătoare vederilor, dar, în plus, pot avea parametri de intrare.

Pentru funcţiile de tip tabel simplu, sintaxa generală are forma:

```
CREATE FUNCTION [<nume_schemă>.]<nume_funcţie>
( [ { <@nume_parametru> [AS] [<tip_nume_schema>.]<tip_date_parametru>
            [=<valoare_implicită>] [ READONLY ]} [ ,...n ] ] )
RETURNS TABLE
 [WITH <opţiune_funcţie> [ ,...n ] ]
 [AS]
RETURN [ ( ]<interogare SELECT> [ ) ]
[ ; ]
```

Definiţia funcţiei (`RETURNS TABLE`) indică faptul că ea returnează un set de date sub forma unui tabel bidimensional. Tabelul (`<interogare SELECT>`) - este însăşi interogare, al cărei rezultat va fi returnat de funcţie.

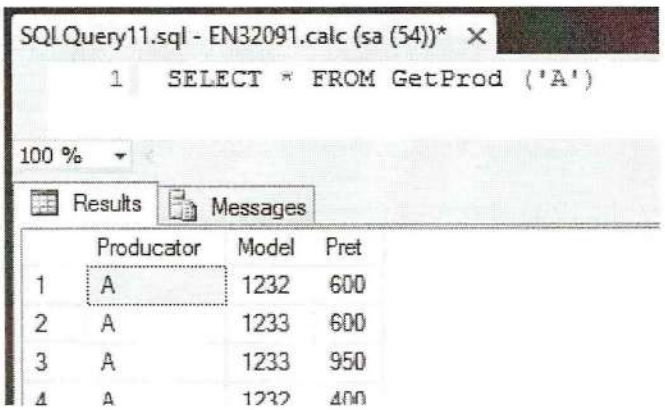
Fie se doreşte să se realizeze o funcţie analogică celei scalare din ultimul exemplu, dar va returna liniile reprezentate de coloanele ***Producator***, ***Model*** şi ***Pret***. Vor fi selectate doar datele specificate de parametrul de intrare a funcţiei. De asemenea, textul funcţiei va fi criptat, pentru ca alţi utilizatori să beneficieze de ea, dar să nu poată citi textul sau corecta.

```
IF OBJECT_ID ('GetProd','FN') IS NOT NULL DROP FUNCTION GetProd
GO
CREATE FUNCTION GetProd (@producator CHAR(1))
    RETURNS TABLE
    WITH ENCRYPTION
    AS
    RETURN
      (
       SELECT p.Producator, pc.Model, pc.Pret
       FROM produse AS p JOIN pc_uri AS pc ON p.Model = pc.Model
       WHERE Producator = @producator
      )
```

O funcţie Inline returnează un tabel ca rezultat, fără a fi nevoie să se definească explicit structura acestuia. Liniile şi coloanele sunt definite de o singură comandă *SELECT*. Cu alte cuvinte, nu sunt permise blocuri *BEGIN ... END*.

Fiecare câmp din interogarea-sursă trebuie să aibă un nume, ceea ce înseamnă că expresiile cu mai mulţi operanzi trebuie sa aibă neapărat un alias. În afară de aceasta, utilizarea clauzei *ORDER BY* este admisă numai dacă se foloseşte împreună cu *TOP* în instrucţiunea *SELECT*.

Apelul unei funcţii de acest tip se poate realiza făcând referire la clauza *FROM* a instrucţiunii *SELECT*. De exemplu:



8.5.3. Funcţii de tip tabel multi-instrucţiune

Funcţiile de tip scalar admit utilizarea oricărui număr de instrucţiuni Transact-SQL, dar întorc un singur rezultat - un scalar. Al doilea tip de funcţii definite de utilizator poate returna un tabel, dar corpul acestuia reprezintă doar o singură interogare. Funcţiile de tip tabel multi-instrucţiune returnează o soluţie care îmbină primele două tehnici, şi anume, ele conţin o mulţime de instrucţii Transact-SQL în calitate de corp, şi întorc un tabel în calitate de rezultat.

Sintaxa acestor funcţii:

```
CREATE FUNCTION [<nume_schemă>.]nume_funcţie
( [ { <@nume_parametru> [AS] [<tip_nume_schema>.]<tip_date_parametru>
            [=<valoare_implicita>] [ READONLY ]} [ ,...n ] ] )
RETURNS <@variabilă_returnată> TABLE <definiţie_tip_tabel>
   [WITH <opţiune_funcţie> [ ,...n ] ]
   [AS]
   BEGIN
     <corp_funcţie>
     RETURN
   END [ ; ]
```

unde:

- `<definiţie_tip_tabel>` - descrierea tabelului (coloane şi constrângeri) care va prezenta rezultatul. Tabelul este, întotdeauna, plasat în grupul de fişiere primare
- `<corp_funcţie>` - în funcţiile de tip tabel multi-instrucţiune, corpul funcţiei reprezintă un set de instrucţiuni Transact-SQL, care populează variabila *TABEL* returnată

Conform sintaxei funcţiei de tip tabel multi-instrucţiune, aceasta se aseamănă foarte mult cu structura funcţiei scalare. Primele trei linii sunt neschimbate. În secţiunea *RETURNS* e declarată o variabilă care este de tip *TABLE*. După aceea, în paranteze, trebuie să fie descrise câmpurile tabelului dezirabil. După cuvântul-cheie *AS*, urmează blocul *BEGIN* şi *END*, între care se poate executa orice număr de instrucţiuni. Blocul se termină cu cuvântul rezervat *RETURN*.

Totuşi, există o deosebire de funcţiile scalare - după *RETURN*, se specifică numele variabilei, a cărei valoare ar trebui să fie rezultatul. În cazul dat, nu trebuie să fie specificat nimic. Declararea variabilei a fost deja efectuată în secţiunea *RETURNS* şi descris formatul acestei variabile. Corpul funcţiei poate şi trebuie să umple această variabilă cu valori şi inclusă în rezultat.

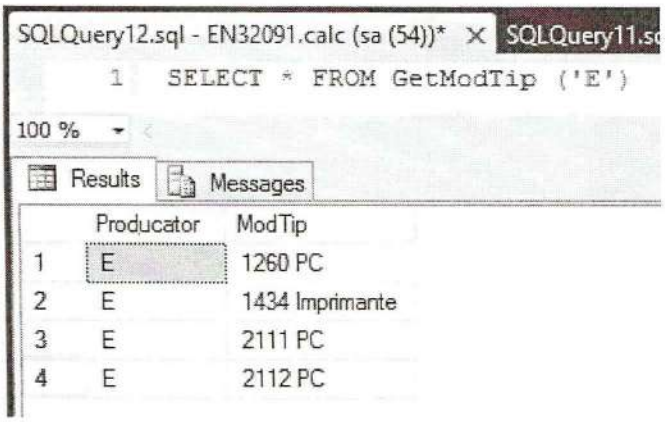
Acum, se ia în considerare un exemplu, care poate fi executat doar cu funcţie de tip tabel multi-instrucţiune şi nu poate fi rulat cu funcţie Inline. Funcţiile multi-instrucţiune returnează un tabel rezultat, fiind necesară şi definirea structurii acestuia. Această categorie poate include crearea de secvenţe de cod mult mai sofisticate.

```
IF OBJECT_ID ('GetModTip','FN') IS NOT NULL DROP FUNCTION GetModTip
```

```
GO
CREATE FUNCTION GetModTip (@Prod CHAR(1))
      RETURNS @ret TABLE
        (Producator CHAR(1), ModTip VARCHAR(15))
      AS
       BEGIN
         INSERT @ret
             SELECT Producator, Model+' '+Tip
                FROM produse
                WHERE Producator =@Prod;
         RETURN
       END
```

În exemplul de faţă, în calitate de rezultat este declarată variabila ***@ret***, care reprezintă un tabel cu două câmpuri: ***Producator*** de tip *CHAR(1)* şi ***ModTip*** de tip *VARCHAR* cu lungimea de 15 caractere. În corpul funcţiei, în acest tabel, se înregistrează valori din tabelul ***produse*** şi, în final, se execută operatorul *RETURN*. Funcţia de tip tabel multi-instrucţiune are numele ***GetModTip*** şi are un parametru.

Executarea funcţiei deloc nu se deosebeşte de apelarea funcţiilor examinate interior. De exemplu, următoarea interogare selectează toate datele care sunt returnată de funcţia:



8.5.4. Modificarea şi suprimarea funcţiilor definite de utilizator

8.5.4.1. Modificarea funcţiilor definite de utilizator

Modificarea unei funcţii este posibilă numai printr-o instrucţiune Transact-SQL DDL - *ALTER FUNCTION*. Într-adevăr, în acest caz, instrucţiunile *CREATE FUNCTION* şi *ALTER FUNCTION* au aproape aceeaşi funcţie, deoarece, în ambele cazuri, întregul corp al funcţiei trebuie să fie rescris în întregime. Nu este posibilă adăugarea sau ştergerea unei singure linii în ea. Singura diferenţă constă în faptul că funcţia este creată sau modificată.

De exemplu, este reprezentată modificarea unei funcţiei create interior:

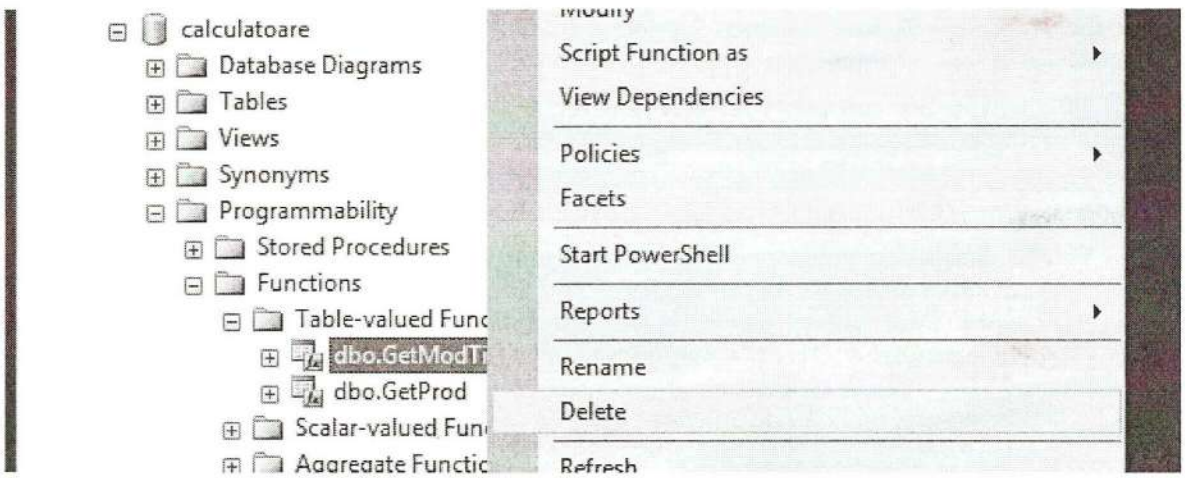
```
ALTER FUNCTION GetProd (@producator CHAR(1))
    RETURNS TABLE
    WITH ENCRYPTION
    AS
    RETURN
     (
      SELECT p.Producator, pc.Model, pc.Pret
      FROM produse AS p JOIN pc_uri pc ON p.Model = pc.Model
      WHERE Producator =@producator
     )
```

Deci, se observă cu uşurinţă că singura modificare sintactică, care există între crearea şi modificarea unei funcţii, este înlocuirea cuvintelor-cheie *CREATE FUNCTION* cu cuvintele-cheie *ALTER FUNCTION*.

Trebuie menţionat că modificarea unei funcţii nu permite schimbarea numelui. În afară de aceasta, nu se poate transforma o funcţie scalară într-una de tip tabel multi-instrucţiune sau invers.

8.5.4.2. Suprimarea funcţiilor definite de utilizator

Suprimarea funcţiei definite de utilizator, ca şi a procedurii stocate sau a oricărui alt obiect al bazei de date, poate fi făcută în două moduri. Prima - este metoda grafică. Această metodă nu va fi detaliată, deoarece este asemănătoare pentru toate obiectele bazei de date. Este suficient să se extindă, în exploratorul de obiect, nodurile corespunzătoare căii spre funcţia dorită, să se aplice un click-dreapta pe ea şi să se selecteze opţiunea *DELETE*



În ce priveşte limbajul Transact-SQL, metoda este aceeaşi ca şi pentru orice alt obiect din baza de date. Astfel, este folosită instrucţiunea *DROP* şi adaptată la cazul unei funcţii. Iată metoda de suprimare a unei funcţii, folosind codul Transact-SQL:

```
DROP FUNCTION { [<nume_schemă>.]<nume_funcţie>} [ ,...n]
```

Instrucţiunea *DROP FUNCTION* realizează ştergerea funcţiei definite de utilizator de orice tip. După această instrucţiune, trebuie să fie specificat doar numele funcţiei.

8.6. Funcţiile definite de utilizator VS proceduri stocate

Funcţiile definite de utilizator şi procedurile stocate sunt obiecte SQL Server, care stochează unul sau mai multe instrucţiuni Transact-SQL sub un singur nume, sub forma unei rutine executabile. Deşi pot fi puse în aplicare aceleaşi funcţionalităţi sau similare, folosind fie o funcţie definită de utilizator, fie o procedură stocată, codul va fi semnificativ diferit în funcţie de tehnica aleasă.

Avantajele folosirii procedurilor stocate sau funcţiilor definite de utilizator sunt următoarele:

- O procedură sau o funcţie odată creată este stocată în baza de date şi poate fi apelată ori de câte ori este necesar.
- Procedurile stocate şi funcţiile definite de utilizator reduc costurile de compilare a codului Transact-SQL, menţinând în cache planurile pentru reutilizare în curs de executare repetată. Prin urmare, acestea nu trebuie să fie reanalizate şi reoptimizate pentru fiecare utilizare, reduc semnificativ timpii de execuţie.
- Elementele de control ale unei proceduri sau funcţii sunt executate ca un singur lot de coduri. Acest lucru poate reduce considerabil traficul de reţea între server şi client, deoarece este trimis prin reţea numai apelul pentru rularea procedurii sau funcţiei.

Principalele diferenţe dintre procedurile stocate şi funcţiile definite de utilizator [8]:

- O funcţie definită de utilizator trebuie să returneze un rezultat constituit dintr-un singur set. O procedură stocată poate, la rândul său returna o valoare sau chiar mai multe seturi de rezultate sau chiar să nu returneze nimic.
- O funcţie definită de utilizator e apelată ca o expresie şi acceptă numai parametri de intrare. Procedurile stocate acceptă parametrii de intrare de la şi pot returna valori prin parametri de ieşire către un program apelant.
- O funcţie definită de utilizator poate fi apelată din procedură, în timp ce o procedură nu poate fi apelată din funcţia definită de utilizator.
- O funcţie definită de utilizator nu poate fi folosită în unele funcţii non-deterministice (care returnează un rezultat diferit cu acelaşi parametru de intrare), precum *NEWID*(), *NEWSEQUENTIALID()*, *RAND*() sau *TEXTPTR()*, în timp ce o procedură stocată poate utiliza aceste funcţii.
- O funcţie definită de utilizator opreşte întotdeauna executarea codului, în caz de eroare, în timp ce, cu o procedură stocată, poate continua până la următoarea instrucţiune, folosind codul de gestiune a erorii corespunzător.
- O instrucţiune *SET* nu poate fi autorizată într-o funcţie definită de utilizator, în timp ce o procedură stocată - poate.
- O funcţie definită de utilizator nu poate fi autorizată de clauza *FOR XML*. Rezultatul *FOR XML* poate fi folosit într-o procedură stocată.
- O funcţie definită de utilizator nu admite executarea acţiunilor care modifică starea bazei de date, în schimb o procedură stocată - poate.

Întrebări de control:

1. Proceduri stocate, facilităţi de utilizare.
2. Clasificarea procedurilor stocate.
3. Structura procedurilor stocate. Parametri de intrare şi de ieşire.
4. Metodele de creare şi gestionare a procedurilor stocate
5. Rolul funcţiei definită de utilizator de tip scalar.
6. Caracteristicile funcţiei definite de utilizator de tip tabel simplu.
7. Instrucţiunile de modificare şi suprimare ale funcţiilor definite de utilizator.
8. Principalele diferenţe dintre procedura stocată şi funcţia definită de utilizator.

Sarcini practice:

1. Să se creeze proceduri stocate în baza exerciţiilor din capitolul 4 .Parametrii de intrare trebuie să corespundă criteriilor din clauzele *WHERE* ale exerciţiilor respective.
2. Să se creeze o procedură stocată care ar elimina toate produsele unui producător sau unul din tipurile de produse ale acestui producător. În calitate de parametru de intrare, să se ia numele producătorului şi tipul produsului.
3. Să se creeze o procedură stocată care ar insera în baza de date un model nou de imprimantă. În cazul în care datele inserate sunt incorecte sau incomplete, să se afişeze un mesaj de avertizare. În calitate de parametri de intrare apar datele pentru modelul respectiv.
4. Să se creeze o procedură stocată care, în calitate de parametru de intrare, să aibă numărul de tipuri de produse fabricate de un producător. În urma executării procedurii, trebuie să se afişeze un mesaj informativ, care să includă valoarea parametrului inserat şi un tabel cu coloanele ***Producator***, ***Produs***, ***Nr_ De_Modele*** pentru fiecare tip de produs.

5. Să se creeze o procedură stocată care ar calcula prețul ce trebuie să-l achite cumpărătorul în timpul promoțiilor speciale la magazinul de calculatoare. Procedura va trebui să calculeze prețul nou pentru toate produsele unui producător anumit, în cazul când se anunță promoția. Un cumpărător poate procura deodată un singur model de produse, dar mai multe unități de acest model. Reducerea poate fi de câteva tipuri în funcție de tipul produsului, prețul inițial și numărul de unități de marfă procurată.

   În calitate de parametri de intrare, se vor lua:

   - modelul produsului procurat de cumpărător;
   - numărul de unități de marfă procurată;
   - procentul de reducere la preț în funcție de numărul de unități de marfă procurată;
   - procentul de reducere la preț în dependență de tipul de marfă procurată;
   - procentul de reducere la preț în dependență de prețul produsului fără reducere.

   Astfel, trebuie să se afișeze ***Modelul***, ***Tipul produsului***, ***Producătorul***, ***Prețul fără reducere***, ***Prețul cu reducere*** și ***Suma totală spre achitare***. ***Suma totală spre achitare*** trebuie să țină cont de reducerile calculate. Implicit asupra produselor procurate nu trebuie sa influențeze nici o reducere.

6. Să se creeze funcții definite de utilizator în baza exercițiilor din capitolul 4. Parametrii de intrare trebuie să corespundă criteriilor din clauzele *WHERE* ale exercițiilor respective.

7. Să se scrie funcția care ar calcula diferența aritmetică dintre valoarea curentă a înregistrării în câmpul ***Pret*** al tabelului ***pc_uri*** și valoare câmpului precedent. Ordonarea înregistrărilor în tabel sa fie făcută după câmpul ***Cod***.

   În calitate de exemplu al utilizării acestei funcții poate servi câmpul ***diferența*** din screenshot-ul de mai jos. Inițial câmpul ***diferența*** = 0.

Results | Messages

	Cod	Model	Viteza	Ram	Hd	Cd	Pret	diferenta
1	1	1232	500	64	5	12x	600	-600
2	2	1121	750	128	14	40x	850	-250
3	3	1233	500	64	5	12x	600	250
4	4	1121	600	128	14	40x	850	-250
5	5	1121	600	128	8	40x	850	0

8. Folosind tabelul ***laptop_uri***, să se creeze o funcție definită de utilizator, care ar citi numele câmpului și ar ataşa sufixul respectiv la valoare necesară. De exemplu, în cazul în care câmpul este ***Pret***, se adaugă sufixul "$", iar dacă câmpul este ***Viteza***, se ataşează sufixul "GHz" etc.

   Să se analizeze screenshot-ul de mai jos în calitate de model:

	Cod	Model	Viteza	Ram	Hd	Pret	Ecran
1	1	1298	3.5 GHz	32 GB	4 GB	700.00$	11"
2	2	1321	5.0 GHz	64 GB	8 GB	970.00$	12"
3	3	1750	7.5 GHz	128 GB	12 GB	1200.00$	14"

   Formatul funcției este următor:

```
nume_funcție(nume_câmp)
```

9. Se dorește realizarea unei funcții definite de utilizator din baza de date calculatoare, care ar putea transforma câmpul ***Pret*** din dolari (starea curentă) în euro sau lei. Se utilizează tabelul ***imprimante***. Formatul funcției este următorul:

   `nume_functie(valoare_pret, curs_BNM, valuta).`

   Valoarea afișată trebuie să fie de tip *MONEY*.

10. Să ce creeze o funcție definită de utilizator, care ar returna toate datele de un anumit produs din stoc. Se definește următorul format al funcției:

    `nume_funcție(preț_minimal, pret_maximal, tip_produs)`

    unde parametrii pot lua valori

    `tip_produs` – *pc*, *laptop* sau *imprimanta*,

    `pret_minimal` - 0.00, valoarea implicită,

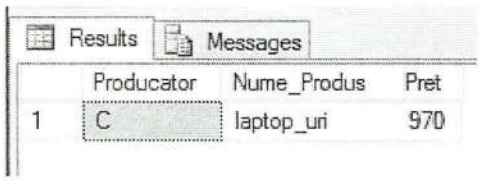
    `pret_maximal` - 10000.00, valoarea implicită.

11. Se cere realizarea unei funcții definite de utilizator, care ar calcula cel mai ieftin sau cel mai scump produs al unui producător. Producătorul este luat în calitate de parametru. Formatul funcției este următorul:

    `nume_funcție(nume_producător, flag)`

    unde, dacă `flag = 1`, va afișa prețul minimal pentru utilizatorul dat, iar `flag = 0` va afișa prețul maximal.

    Un exemplu de rezultate returnate de funcție:



9. CREAREA ŞI UTILIZAREA DECLANŞATOARELOR

Capitolul este dedicat gestionării şi executării declanşatoarelor de către SQL Server 2014. În particular, sunt analizate declanşatoarele de tip *DML*, *DDL* şi *LOGON*, descrise structura declanşatoarelor, precum şi tratată problematica cu privire la crearea şi executarea declanşatoarelor.

Microsoft SQL Server 2014 oferă două mecanisme principale pentru implementarea a ceea ce în aplicaţiile de baze de date poartă numele de *business rules:* constrângerile şi declanşatoarele. Termenul se referă la acele reguli, restricţii, care ţin in mod inerent de structura bazei de date şi sunt induse de semantica unei colecţii de date, fiind comune tuturor aplicaţiilor care folosesc această bază de date şi independente de fiecare aplicaţie în parte.

9.1. Clasificarea tipologică şi structura declanşatoarelor

Declanşatoarele sunt o clasă specială de proceduri stocate, asociate unui tabel sau unei baze de date, sau unei instanţe SQL Server, definite pentru a fi lansate în execuţie, în mod automat, la iniţierea unui eveniment şi reprezintă un instrument puternic pentru elaborarea bazelor de date active.

Aria de utilizare a declanşatoarelor este cât se poate de extinsă: implementarea constrângerilor complexe, asigurarea integrităţii bazei de date, managementul securităţii datelor, respectarea integrităţii referenţiale între nodurile unei baze de date distribuite, jurnalizarea operaţiunilor (tranzacţiilor), sincronizarea replicilor unei baze de date etc.

SQL Server oferă trei tipuri generale de declanşatoare: *DML* şi *DDL* şi *LOGON.*

9.1.1. Declanşatoare DML

Declanşatoarele DML sunt lansate atunci când în baza de date are loc un eveniment al limbajului de manipulare a datelor. Evenimentele DML sunt reprezentate instrucţiunile *INSERT*, *UPDATE* sau *DELETE*, care actualizează datele unui tabel sau viziuni. Un declanşator DML poate interoga alte tabele şi poate conţine instrucţiuni Transact-SQL complexe. Atât declanşatorul, cât şi instrucţiunea care l-a lansat, sunt tratate ca o singură tranzacţie care poate fi anulată, în caz de eşec, şi atunci, în acest declanşator, sunt restabilite schimbările. Dacă este detectată vreo eroare a serverului (de exemplu, insuficienţă de spaţiu pe disc), tranzacţia este anulată iar starea bazei de date de până la lansarea tranzacţiei este restabilită.

Declanşatoarele pot modifica în cascadă tabelele din baza de date, însă aceste modificări pot fi executate mai eficient, utilizând constrângerile de integritate referenţiale în cascadă.

În schimb, declanşatoarele pot detecta operaţiile *INSERT*, *UPDATE* şi *DELETE* incorecte sau rele cu premeditare şi să impună alte restricţii mai complexe, decât cele ce pot fi definite cu constrângerea *CHECK.*

De exemplu, un declanşator poate utiliza un *SELECT* definit asupra altui tabel, pentru a compara cu datele inserate sau modificate şi pentru a executa acţiuni adiţionale, cum ar fi modificarea datelor sau afişarea unui mesaj de eroare definit de utilizator. De asemenea, declanşatoarele pot evalua starea tabelului înainte şi după modificare şi realiza acţiuni bazate pe deosebirea detectată.

Diverse declanşatoare DML de acelaşi tip (*INSERT*, *UPDATE* sau *DELETE*) definite asupra aceluiaşi tabel pot executa acţiuni diferite ca răspuns la aceleaşi instrucţiuni de actualizare.

Declanşatoarele sunt utile, mai ales, în crearea constrângerilor şi consistenţelor de acces la baza de date. Fie, de exemplu, un program de control al stocului. Poate fi elaborat un declanşator care reacţionează la evenimentul că, dacă un anumit produs atinge cantitatea minimă necesară în stoc, atunci este lansată o solicitare de cumpărare a acestui produs, folosind un raport.

Pot fi menţionate două din aplicaţiile principale ale declanşatoarelor:

- Elaborarea mecanismelor de validare care atrag după sine căutări în mai multe tabele.
- Actualizarea altor tabele în funcţie de includerea datelor sau modificarea tabelului curent.

Pot fi programate următoarele tipuri de declanşatoare DML:

- **Declanşatoare *AFTER*.** Declanşatoarele *AFTER* sunt lansate după executarea reuşită a instrucţiunilor *INSERT*, *UPDATE* sau *DELETE*. Specificarea *AFTER* este identică cu specificarea *FOR*, care este singura opţiune disponibilă în versiunile mai vechi ale ***Microsoft SQL Server.*** Declanşatoarele *AFTER* pot fi definite numai asupra unui singur tabel.
- **Declanşatoare *INSTEAD OF.*** Declanşatoarele *INSTEAD OF* sunt executate în locul acţiunii uzuale de declanşare. Declanşatoarele de acest tip pot fi, de asemenea, definite asupra viziunilor, care, la rândul lor, sunt construite din mai multe tabele. Acestea pot extinde tipurile de actualizări pe care o viziune poate să le suporte.
- **Declanşatoare *CLR.*** Un declanşator *CLR* poate fi sau un declanşator *AFTER*, sau un declanşator *INSTEAD OF*. Un declanşator *CLR* poate fi, de asemenea, un declanşator de tipul DLL. În locul executării unei proceduri stocate, un declanşator *CLR* execută una sau mai multe metode scrise în cod controlat, care sunt parte a unei mulţimi create în ***.NET Framework*** şi încărcate în ***SQL Server.***

Instrucţiunile declanşatoarelor DML folosesc două tabele speciale: tabelul ***deleted*** şi tabelul ***inserted***. Sistemul SQL Server creează şi gestionează aceste tabele în mod automat. Tabelele menţionate sunt temporare, rezidente în memorie şi pot fi utilizate pentru a testa efectele unor modificări de date şi pentru a seta condiţii asupra acţiunilor declanşatoarelor de tip DML. Nu este posibilă modificarea directă a datelor în aceste tabele sau efectuarea instrucţiunilor DDL ca, de exemplu, *CREATE INDEX*.

Astfel, tabelul ***deleted*** conţine linii în starea de până la aplicare instrucţiunii DML, iar tabelul ***inserted*** – în starea de după aplicarea instrucţiunii DML.

Instrucţiune DML	**Tabelul *inserted***	**Tabelul *deleted***
INSERT	Linii inserate	Vid
UPDATE	Linii de după modificare	Linii de până la modificare
DELETE	Vid	Linii ce trebuie şterse

Trebuie menţionat că în declanşatoarele de tip DML, tabelele ***deleted*** şi ***inserted*** sunt, în primul rând, folosite pentru a efectua:

- Extinderea integrităţii referenţiale dintre tabele.
- Inserarea sau actualizarea datelor în tabelele utilizate de o viziune.
- Testarea la erori şi tratarea acestora.
- Depistarea diferenţei dintre starea tabelului, înainte şi după modificarea datelor şi aplicarea unor acţiuni bazate pe diferenţa detectată.

9.1.2. Declanşatoare DDL

Declanşatoarele DDL au apărut prima dată în *SQL Server 2005* şi se utilizează pentru controlul evenimentelor la nivel de bază de date sau la nivel de server. Aceste declanşatoare sunt invocate, dacă au loc în baza de date sau pe server evenimente ale limbajului de definire a datelor (DDL). Declanşatoarele DDL, asemenea declanşatoarelor regulare, lansează o procedură stocată drept răspuns la un eveniment.

Însă, spre deosebire de declanşatoarele DML, acestea nu sunt lansate ca răspuns la instrucţiuni de tipul *UPDATE*, *INSERT* sau *DELETE* aplicate asupra tabelului sau viziunii. În schimb, sunt puse în funcţiune de varietatea de instrucţiuni de definire a datelor. Aceste instrucţiuni sunt cele care încep cu cuvintele rezervate *CREATE*, *ALTER*, *DROP*, *GRANT*, *DENY*, *REVOKE* sau *UPDATE STATISTICS*. Declanşatoarele DDL pot fi utilizate pentru administrarea sarcinilor, precum cele de audit şi reglare a operaţiilor asupra bazei de date sau server.

Declanşatoarele DDL sunt utilizate atunci când:

- Se cere prevenită orice modificare a schemei bazei de date.
- Se cere înregistrarea modificărilor sau evenimentelor ce au avut loc în schema bazei de date.
- Se doreşte să se producă ceva în baza de date, ca răspuns la modificarea schemei bazei de date.

Declanşatoarele DDL sunt puse în funcţiune numai după ce este executată vreo instrucţiune de definire a datelor, adică acestea nu pot fi declanşatoare de tipul *INSTEAD OF*. Prin urmare, declanşatoarele DDL se lansează drept răspuns la un eveniment specificat de sintaxa limbajului de definire a datelor al Transact-SQL. În plus, declanşatoarele DDL reacţionează numai la evenimentele care se petrec în baza de date curentă sau pe serverul curent.

Trebuie menţionat că declanşatoarele DDL nu utilizează tabelele speciale *inserted* sau *deleted*. În afară de aceasta, pentru un eveniment DDL pot fi create mai multe declanşatoare.

9.1.3. Declanşatoare Logon

Declanşatoarele *Logon* sunt proceduri stocate care se lansează ca răspuns la un eveniment *LOGON*. Acest eveniment apare atunci când este stabilită o sesiune utilizator cu o instanţă SQL Server. Declanşatoarele *Logon* se lansează după ce faza de autentificare a logării este în final, dar înainte de stabilirea sesiunii utilizatorului. Prin urmare, toate mesajele care provin din interiorul declanşatorului care în mod obişnuit le primeşte utilizatorul, cum ar fi mesajele de eroare sau mesajele de la instrucţiunea *PRINT*, sunt îndreptate în jurnalul de erori al sistemului SQL Server. Declanşatoarele *Logon* nu se lansează, dacă autentificarea eşuează.

Declanşatoarele *Logon* sunt folosite pentru auditul şi controlul sesiunilor serverului, cum ar fi urmărirea activităţii de autentificare, restricţionarea accesului la sistem, sau limitarea numărului de sesiuni pentru un anumit utilizator.

Evenimentul *LOGON* corespunde evenimentului *AUDIT_LOGIN SQL Trace*, care pot fi folosite în notificările de evenimente. Principala diferenţă dintre declanşatoare şi notificările de evenimente constă în faptul că declanşatoarele se lansează sincron cu evenimentele, în timp ce notificările se lansează asincron. Acest lucru înseamnă că, dacă se doreşte oprirea unei sesiuni deja stabilite, trebuie folosit un declanşator *Logon*. O notificare de evenimentul *AUDIT_LOGIN* nu poate fi folosită în acest scop.

9.1.4. Structura declanşatoarelor

Un declanşator este compus din două părţi:

- Instrucţiuni SQL care lansează declanşatorul: *INSERT*, *DELETE* şi *UPDATE* pentru un tabel sau viziune în cazul declanşatorului DML sau *CREATE, ALTER, DROP, GRANT, DENY, REVOKE* sau *UPDATE STATISTICS* pentru un server întreg sau bază de date în cazul declanşatorului DDL.
- Acţiunea exercitată de declanşator: în realitate, este un bloc de instrucţiuni SQL, care vor fi executate de declanşator.

Unele instrucţiuni nu pot fi utilizate de declanşatorul DML. Acestea sunt afişate mai jos. O parte din ele, fiind necunoscute, vor fi tratate mai târziu:

ALTER DATABASE	*CREATE DATABASE*	*DROP DATABASE*
RESTORE DATABASE	*RESTORE LOG*	*RECONFIGURE*

În plus, următoarele instrucţiuni ale Transact-SQL nu sunt permise în corpul declanşatorului DML, atunci când este utilizat asupra tabelului sau viziunii care constituie şi scopul acţiunii declanşatorului:

CREATE INDEX (inclusiv *CREATE SPATIAL INDEX* şi *CREATE XML INDEX*)	*ALTER INDEX*	*DROP INDEX*
DBCC DBREINDEX	*ALTER PARTITION FUNCTION*	*DROP TABLE*
ALTER TABLE când e utilizată pentru: • Adăugarea , modificarea sau suprimarea coloanelor. • Comutarea partiţiilor. • Adăugarea sau suprimarea constrângerilor *PRIMARY KEY* sau *UNIQUE*		

Pentru crearea declanşatoarelor în limbajul Transact-SQL, se folosesc trei tipuri de instrucţiuni.

Declanşatoarele pentru controlul actualizărilor (declanşatoarele DML) sunt bazate pe instrucţiunile definite asupra unui tabel sau viziuni.

```
CREATE TRIGGER [nume_chema.] nume_declansator
  ON { tabel | viziune }
  [ WITH <opţiune_declanşator_dml > [ ,...n ] ]
  { FOR | AFTER | INSTEAD OF }
  { [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
  [ NOT FOR REPLICATION ]
  AS { instructiune_sql  [ ; ] [ ,...n ]
     | EXTERNAL NAME <specificator metoda [ ; ] > }
<opţiune_dml_declansator> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clauza ]
<specificator_metoda> ::=
    nume_bloc.nume_clasa.nume_metoda
```

Declanşatorul DDL apelează la instrucţiunile *CREATE, ALTER, DROP, GRANT, DENY, REVOKE, UPDATE STATISTICS* şi are următoarea structură generală:

```
CREATE TRIGGER nume_declansator
  ON { ALL SERVER | DATABASE }
  [ WITH <optiune_declansator_ddl> [ ,...n ] ]
  { FOR | AFTER } {tip_eveniment | grup_eveniment} [ ,...n ]
  AS {instructiune_sql [ ; ] [ ,...n ]
       | EXTERNAL NAME <specificator metoda>  [ ; ] }
<optiune_ddl_trigger> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clauza ]
```

Sintaxa pentru crearea unui declanşator sistem (*Logon*) este următoarea:

```
CREATE TRIGGER nume_declanşator
  ON ALL SERVER
  [ WITH <optiune_declansator_logon> [ ,...n ] ]
  { FOR | AFTER } LOGON
  AS {instructiune_sql [ ; ] [ ,...n ]
       | EXTERNAL NAME <specificator metoda>  [ ; ] }
<optiune_declanşator_logon> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clauza ]
```

În continuare, sunt propuse unele explicaţii ale termenilor utilizaţi în descrierea structurilor declanşatoarelor.

`nume_schema`

Este numele schemei la care se referă declanşatorul DML. Declanşatoarele DML sunt orientate spre schema tabelului sau viziunii pentru care acestea sunt create. Numele schemei nu poate fi specificat pentru declanşatoarele de tip DDL sau LOGON.

`nume_declansator`

Numele declanşatorului trebuie să fie definit în conformitate cu normele de identificare. Unica constrângere suplimentară constă în faptul că un nume de declanşator nu poate începe cu # sau # #.

`tabel | viziune`

Reprezintă tabelul sau viziunea pe care e definit declanşatorul DML. Specificarea numelui complet al tabelului sau viziunii este opţională. O viziune poate fi referită numai de către un declanşator de tip *INSTEAD OF*. Declanşatoarele DML nu pot fi definite pe un tabel temporar local sau global.

`DATABASE`

Se defineşte în calitate de domeniu de aplicare a unui declanşator DDL. Reprezintă numele bazei de date curente. Dacă numele este specificat, declanşatorul este lansat întotdeauna când în baza de date curentă au loc evenimentele ***tip_eveniment*** sau ***grup_eveniment***.

`ALL SERVER`

Se defineşte în calitate de domeniu de aplicare a unui declanşator DDL sau *LOGON*. Reprezintă numele serverului curent. Dacă numele se specifică, declanşatorul se lansează întotdeauna când pe serverul curent au loc evenimentele ***tip_eveniment*** sau ***grup_eveniment***.

`WITH ENCRYPTION`

Se criptează textul instrucţiunii de creare a declanşatorului. Utilizarea clauzei *WITH ENCRYPTION* previne faptul ca declanşatorul să fie publicat ca parte a unei replicări *SQL Server*. Această opţiune nu poate fi specificată pentru declanşatoare CLR

`EXECUTE AS`

Specifică contextul de securitate în care este executat declanşatorul. Permite determinarea contului utilizatorului (pe instanţa SQL Server) utilizat pentru validarea permisiunilor asupra oricărui obiect al bazei de date, la care se referă declanşatorul.

`FOR | AFTER`

AFTER specifică faptul că declanşatorul DML este executat doar când toate operaţiile din enunţul SQL al declanşatorului au fost executate cu succes. Toate acţiunile referenţiale în cascadă şi verificările de constrângeri, de asemenea, trebuie să reuşească înainte de executarea acestui trigger.

Opţiunea *AFTER* este implicită în cazul când *FOR* este singurul cuvânt–cheie specificat. Adică. folosirea clauzei *AFTER* este identică cu folosirea clauzei *FOR* utilizată mai mult în ultimele versiuni de SQL Server. În plus, declanşatoarele *AFTER* nu pot fi definite pentru viziuni.

`INSTEAD OF`

Specifică faptul că declanşatorul DML este executat în locul instrucţiunilor de actualizare specificate. Opţiunea *INSTEAD OF* nu poate fi specificată pentru declanşatoarele de tip DDL sau *LOGON*.

Asupra unui tabel sau viziuni poate fi definit cel mult un declanşator *INSTEAD OF* pentru instrucţiunile *INSERT*, *UPDATE*, sau *DELETE*. Cu toate acestea, este posibilă definirea viziunilor asupra viziunilor, unde fiecare din acestea să posede propriul declanşator cu opţiunea *INSTEAD OF*.

Declanşatoarele *INSTEAD OF* nu pot fi create asupra viziunilor actualizabile care utilizează clauza *WITH CHECK OPTION*. În acest caz, sistemul lansează un mesaj de eroare. Utilizatorul trebuie să înlăture opţiunea *WITH CHECK OPTION*, apelând la instrucţiunea *ALTER VIEW* înainte de a defini pe viziunea dată un declanşator *INSTEAD OF*.

`{ [DELETE] [,] [INSERT] [,] [UPDATE] }`

Specifică instrucţiunile de actualizare a datelor care provoacă lansarea declanşatorului. Cel puţin o opţiune din aceste trei trebuie să fie specificată. Se permite să fie incluse în definiţia declanşatorului orice combinaţie din aceste trei operaţii şi în orice ordine.

Pentru declanşatoarele *INSTEAD OF*, opţiunea *DELETE* nu este permisă asupra tabelelor care au legături referenţiale constrânse de acţiunea de ştergere în cascadă referinţă specificând o acţiune *DELETE* cascadă (*cascade action ON DELETE*). În mod similar, nu sunt admise declanşatoare *INSTEAD OF* cu opţiunea *UPDATE* definite asupra tabelelor care au legături referenţiale cu specificaţia de modificare în cascadă (*cascade action ON UPDATE*).

`tip_eveniment`

Este numele unui eveniment Transact-SQL, care, după executare, cauzează lansarea unui declanşator DDL.

`grup_eveniment`

Este numele unui grup de evenimente. Declanşatoarele DDL sunt lansate după apariţia oricărui eveniment ce face parte din grupul de evenimente specificat.

După executarea instrucţiunii *CREATE TRIGGER*, ***grup_eveniment*** acţionează, de asemenea, ca un macrou, prin adăugarea tipurilor de evenimente ce fac parte din viziunea de sistem ***sys.trigger_events***.

`NOT FOR REPLICATION`

Indică faptul că declanşatorul nu trebuie să fie executat în cazul în care vreun agent de replicare modifică tabelul implicat în declanşator.

`instrucţiune_sql`

Reprezintă condiţiile şi acţiunile declanşatorului. Condiţiile sunt nişte criterii adiţionale, care trebuie verificate pentru a fi lansat declanşatorul în urma apariţiei unui eveniment DML, DDL sau LOGON.

`specificator_metoda`

Se aplică la: SQL Server 2008 prin SQL Server 2014.

Se utilizează pentru declanşatorul CLR şi specifică metoda de asamblare a legăturilor cu declanşatorul. Metoda nu trebuie să ia nici un argument şi trebuie să returneze *void*. Argumentul *nume_clasa* trebuie să fie un identificator valid în sistem şi trebuie să existe ca o clasă cu acest nume vizibilă în ansamblu În cazul în care clasa are un nume cu utilizare a punctului pentru separarea spaţiului de nume, atunci numele se ia în paranteze pătrate, [], sau în ghilimele, "". Clasa nu poate fi una imbricată.

9.2. Crearea, activarea şi dezactivarea declanşatoarelor

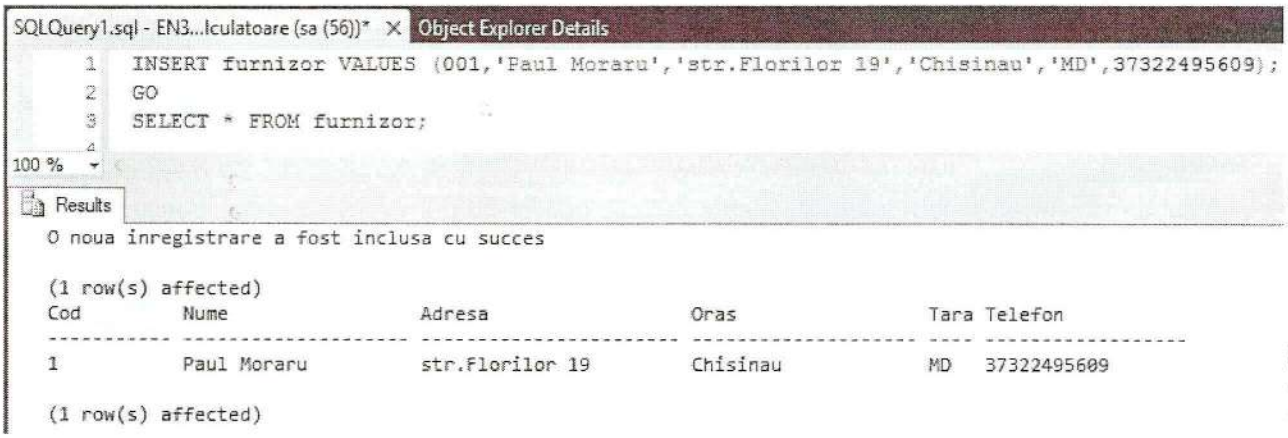
Pentru crearea declanşatoarelor în MS SQL Server 2014, se utilizează mediul ***Transact-SQL Query Editor***. În continuare, sunt examinate câteva exemple de creare a declanşatoarelor.

9.2.1. Crearea declanşatoarelor DML

Se prezintă un declanşator DML, care generează un mesaj de informare. În acest declanşator, se specifică faptul că pentru tabelul ***furnizor***, la includerea unei înregistrări noi, va fi generat un mesaj *"O nouă înregistrare a fost inclusă cu succes"*. Cu acest scop, după clauza *AS*, se scrie comanda *PRINT* şi mesajul respectiv.

```
USE calculatoare
GO
IF OBJECT_ID ('inregistrare_noua', 'TR') IS NOT NULL
   DROP TRIGGER inregistrare_noua;
GO
CREATE TRIGGER inregistrare_noua ON furnizor
   AFTER INSERT
   AS PRINT 'O noua inregistrare a fost inclusa cu succes';
GO
```

Acum se poate scrie o instrucţiune de control, care inserează în tabelul ***furnizor*** o înregistrare nouă şi verifică rezultatul lucrului declanşatorului. În plus, se cere lista tuturor înregistrărilor tabelului în cauză.



Să se observe că, independent de structura instrucţiunilor, care se utilizează, de acum înainte, întotdeauna, când va fi utilizată instrucţiunea *INSERT* asupra tabelului ***furnizor***, va fi afişat mesajul *"O nouă înregistrare a fost inclusă cu succes"*.

În exemplul următor, este creat un declanşator care nu permite ştergerea completă a furnizorilor cu reşedinţa în Moldova, adică ***Tara***='MD'. Declanşatorul respectiv poate fi activat după operaţiile *UPDATE* şi *DELETE* în cazul dacă numărul de înregistrări în tabel a furnizorilor pentru care ***Tara***='MD' este mai mic de 1, apare un mesaj de avertizare, iar operaţiile ce au provocat lansarea declanşatorului sunt derulate înapoi.

În final, se execută o instrucţiune de control *DELETE* şi se obţine mesajul respectiv.

SQLQuery1.sql - EN3...lculatoare (sa (56))* | Object Explorer Details

```
1  USE calculatoare
2  GO
3  IF OBJECT_ID ('furnizori_MD', 'TR') IS NOT NULL DROP TRIGGER furnizori_MD;
4  GO
5  CREATE TRIGGER furnizori_MD ON furnizor
6   AFTER UPDATE,DELETE
7   AS
8   DECLARE @Cod1 varchar(4)
9    SELECT @Cod1=Cod FROM furnizor WHERE Tara='MD'
10    IF COUNT(@Cod1)<1
11    BEGIN
12        RAISERROR ('Atentie pericol de a pierde furnizori din MOLDOVA', 16, 1)
13        ROLLBACK TRANSACTION
14    END
15  GO
16
17  DELETE FROM furnizor WHERE Tara='MD'
```

100 %

Results

```
Warning: Null value is eliminated by an aggregate or other SET operation.
Msg 50000, Level 16, State 1, Procedure furnizori_MD, Line 23
Atentie pericol de a pierde furnizori din MOLDOVA
Msg 3609, Level 16, State 1, Line 17
The transaction ended in the trigger. The batch has been aborted.
```

Ca şi alte obiecte ale bazei de date, declanşatoarele DML suportă instrucţiunile *ALTER* şi *DROP*. Pentru modificarea unui declanşator DDL, acesta, mai întâi, trebuie să fie şters şi apoi creat din nou.

Următorul exemplu modifică declanşatorul ***inregistrare_noua***. Declanşatorul utilizează tabelul ***inserted*** pentru crearea raportului despre valorile actualizate ale coloanei ***Adresa*** a tabelului pe care este definit acest declanşator. Funcţia *NOCOUNT* este activată pentru excluderea din fereastra ***Results*** a mesajelor de executare pentru subinstrucţiunile care se afla în corpul declanşatorului. Funcţia *RTRIM* elimina spaţiul liber (gol) din faţa coloanei *Nume*.

SQLQuery1.sql - EN3...lculatoare (sa (56))* | Object Explorer Details

```
1  USE calculatoare
2  GO
3  ALTER TRIGGER inregistrare_noua ON furnizor
4   AFTER UPDATE
5   AS SET NOCOUNT ON
6    IF UPDATE(Adresa)
7     SELECT 'Furnizorul ' +RTRIM(UPPER(inserted.Nume))+
8            ' are adresa noua - '+UPPER(inserted.Adresa)
9          FROM inserted
10 GO
11
12 UPDATE furnizor SET Adresa='str.Alecu Ruso 1' WHERE Nume='Paul Moraru';
```

100 %

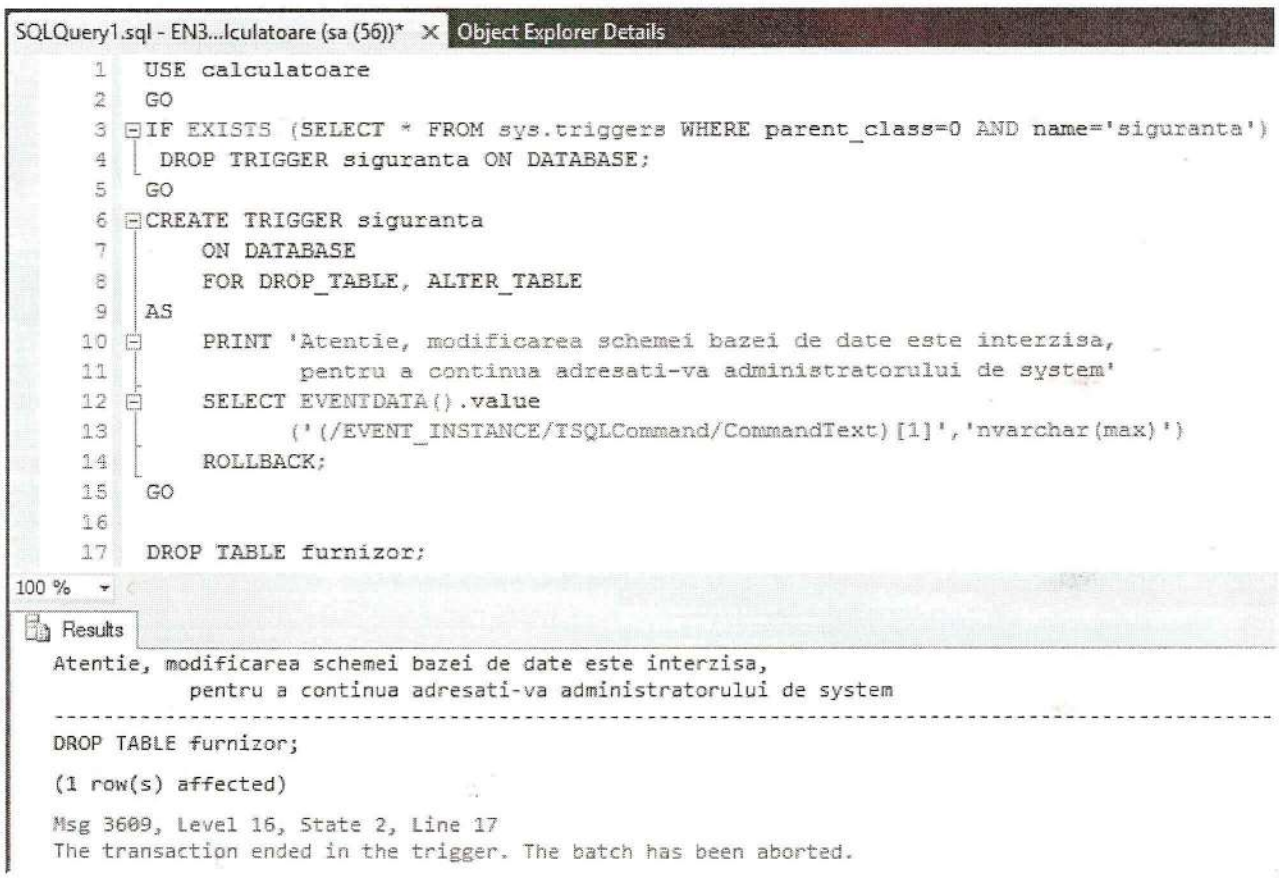
Results

```
-------------------------------------------------------------------------------
Furnizorul PAUL MORARU are adresa noua - STR.ALECU RUSO 1

(1 row(s) affected)
```

9.2.2. Crearea declanşatoarelor DDL

Următorul exemplu ilustrează modul în care un declanşator DDL poate fi utilizat pentru prevenirea modificării schemei unui tabel sau suprimării unui tabel din baza de date:



Obiectele care nu fac parte din domeniul bazei de date, cum ar fi declanşatoarele DDL, nu pot fi solicitate folosind funcţia *OBJECT_ID*. Pentru obiectele care nu se găsesc în viziunea ***sys.objects***, pot fi obţinute numerele de identificare ale obiectelor, folosind interogări asupra viziunilor de sistem respective. Cum se vede din exemplul de mai sus, pentru a obţine numărul de identificare a declanşatorului DDL, se utilizează instrucţiunea *SELECT* asupra ***sys.triggers***.

Pentru a controla funcţionalitatea declanşatorului ***siguranta***, în final, se dă la executare o instrucţiune de ştergere a tabelului ***furnizor***.

Informaţiile despre un eveniment ce lansează un declanşator DDL pot fi capturate, apelând la funcţia *EVENTDATA*. În urma utilizării acestei funcţii, în mesajul apărut, se afişează evenimentul care a lansat declanşatorul respectiv (în cazul nostru *DROP TABLE* ***furnizor***).

9.2.3. Crearea declanşatoarelor LOGON

Urmează un exemplu de declanşator LOGON.

```
USE calculatoare
GO
IF EXISTS (SELECT * FROM master.sys.server_triggers
   WHERE parent_class_desc = 'SERVER' AND name = 'limit_sesiuni')
   DROP TRIGGER limit_sesiuni ON ALL SERVER;
GO
IF EXISTS (SELECT * FROM sys.server_principals WHERE name = 'user1')
   DROP LOGIN user1;
GO
   CREATE LOGIN user1 WITH PASSWORD = 'parola1';
```

```
GO

CREATE TRIGGER limit_sesiuni
   ON ALL SERVER WITH EXECUTE AS 'user1'
   FOR LOGON
AS
   BEGIN
     IF ORIGINAL_LOGIN()= 'user1' AND
     (SELECT COUNT(*) FROM sys.dm_exec_sessions
     WHERE is_user_process = 1 AND original_login_name ='user1')>3
     ROLLBACK;
   END;
```

Instrucţiunile de pe liniile 3 şi 7 verifică dacă, deja, există în sistem vreun declanşator şi vreun login cu un asemenea nume. Pentru aceasta, sunt interogate viziunile de sistem respective şi, dacă asemenea obiecte sunt găsite, acestea sunt suprimate.

Instrucţiunea *CREATE LOGIN* creează un login cu un nume şi parolă definite. Deoarece pentru loginul creat nu sunt explicit specificate drepturile de acces, implicit i se atribuie rolul *PUBLIC*.

Pe linia 13 începe instrucţiunea de creare a declanşatorului care interzice intrarea în sistem a unui utilizator, dacă acesta, deja, are trei sesiuni active create cu acest login pe o instanţă SQL Server. Interogând viziunea ***sys.dm_exec_sessions***, declanşatorul obţine numărul de sesiuni active cu loginul dat. Dacă numarul depăşeşte condiţia, operaţiunea de loghinare este rulată înapoi.

9.2.4. Activarea şi dezactivarea declanşatoarelor

Declanşatoarele pot fi dezactivate sau activate. Activarea unui declanşator nu înseamnă recrearea declanşatorului. Un declanşator dezactivat rămâne în calitate de obiect în baza de date curentă, dar el nu se declanşează. Activarea declanşatorului presupune că acesta va fi lansat, dacă se execută orice tranzacţie pentru care a fost programat.

Pentru a activa sau dezactiva un declanşator, sunt utilizate instrucţiunile *ENABLE TRIGGER* şi *DISABLE TRIGGER*, respectiv.

În continuare, este dezactivat declanşatorul DML ***înregistrare_noua*** care a fost creat pe tabelul ***furnizor*** şi apoi este activat din nou:

```
USE calculatoare
GO
DISABLE TRIGGER inregistrare_noua ON furnizor;
GO
ENABLE TRIGGER inregistrare_noua ON furnizor;
GO
```

Următorul exemplu dezactivează un declanşator DDL la nivelul bazei de date.

```
USE calculatoare
GO
DISABLE TRIGGER siguranta ON DATABASE;
```

Dezactivarea sau activarea unui declanşator DML pot fi realizate şi în cadrul instrucţiunii *ALTER TABLE*, aşa cum este prezentat în exemplul de mai jos:

```
USE calculatoare
GO
ALTER TABLE furnizor DISABLE TRIGGER inregistrare_noua;
```

Exemplul de mai jos dezactivează un declanşator DDL la nivelul unui server:

```
USE calculatoare
GO
DISABLE TRIGGER limit_sesiuni ON ALL SERVER;
```

Următorul exemplu activează toate declanşatoarele DDL la nivelul unui server.

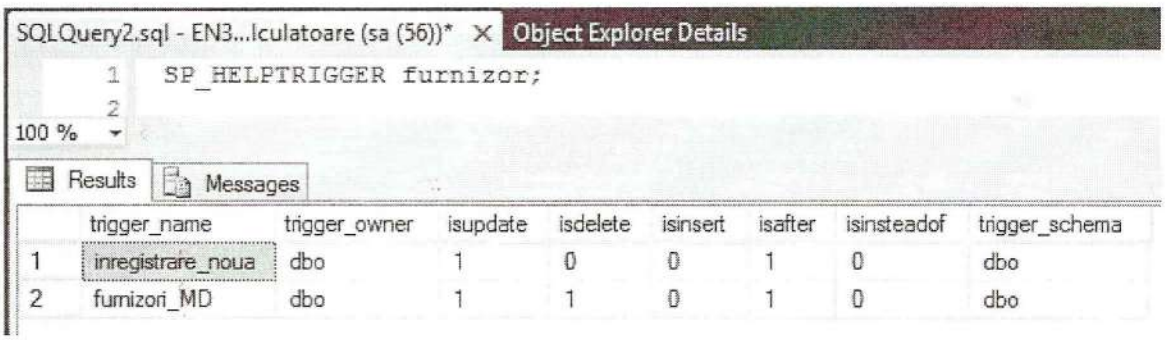
```
USE calculatoare
GO
ENABLE TRIGGER ALL ON ALL SERVER;
```

9.3. Declanşatoarele şi procedurile stocate standard

În mediul ***SQL Server Management Studio*** declanşatoarele la nivel de bază de date se află în dosarul ***Programability*** al ferestrei ***Object Explorer***. Declanşatoarele de acest nivel pot fi uşor programate, dar nu pot fi modificate atât de uşor, ca alte obiecte-program, precum, de exemplu, procedurile stocate. Declanşatoarele la nivel de server se găsesc în dosarul ***Server Objects*** al ferestrei ***Object Explorer***.

Pentru a obţine lista declanşatoarelor ce se află în baza de date curentă este necesar să se scrie interogarea asupra viziunilor ***sys.triggers*** şi ***sys.trigger_events***. Pentru a obţine lista declanşatoarelor la nivel de server, se scriu interogări asupra ***sys.server_triggers*** şi ***sys.server_trigger_events***.

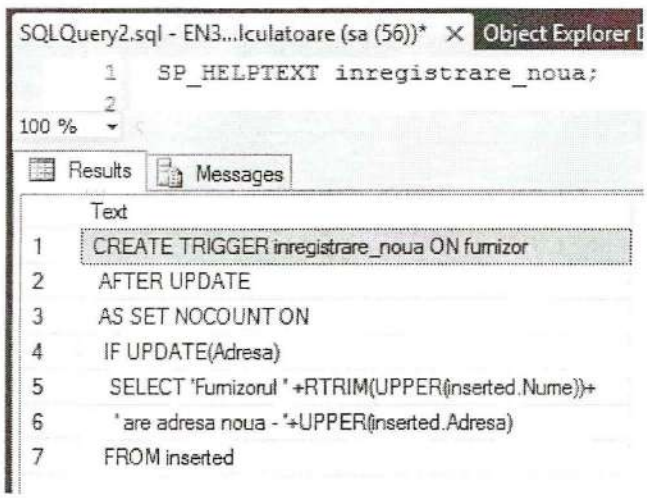
Se pot determina toate declanşatoarele definite asupra unui tabel al bazei de date, folosind o procedură stocată de sistem. Să se observe următoarea instrucţiune, care susţine acest obiectiv:



	trigger_name	trigger_owner	isupdate	isdelete	isinsert	isafter	isinsteadof	trigger_schema
1	inregistrare_noua	dbo	1	0	0	1	0	dbo
2	furnizori_MD	dbo	1	1	0	1	0	dbo

Aplicând procedura stocată **sp_helptrigger**, sunt afişate declanşatoarele tabelului specificat. În acest exemplu, sunt afişate declanşatoarele existente asupra tabelului ***furnizor*** al bazei de date ***calculatoare***.

Acum, fie că se doreşte prezentarea conţinutului unui declanşator DML. Pentru aceasta, se apelează la procedura stocată deja cunoscută din lucrarea practică precedentă:



Aplicând procedura ***sp_helptext***, pot fi observate toate instrucţiunile care formează corpul declanşatorului.

Întrebări de control:

1. Care sunt domeniile de utilizare a declanşatoarelor?
2. Tipurile de declanşatoare DML.
3. Utilizarea tabelelor ***deleted*** şi ***inserted***
4. Declanşatoarele DDL. Care sunt caracteristicile principale ale acestor declanşatoare?
5. Declanşatoarele LOGON. Avantajele utilizării.
6. Crearea declanşatoarelor în Transact –SQL, sintaxa de bază.
7. Activarea - dezactivarea declanşatoarelor în Transact –SQL. Utilizare şi sintaxa de bază.
8. Cum pot fi extrase datele despre declanşatoare din catalogul de sistem?

Sarcini practice:

1. Să se modifice declanşatorul ***inregistrare_noua*** în aşa fel încât în cazul actualizării adresei furnizorului să apară mesajul de informare, care va afişa numele furnizorului, adresa veche şi adresa noua.
2. Să se creeze declanşatoarele care ar insera corect (consecutiv) datele în baza de date ***calculatoare*** şi ar permite evitarea erorilor la nivelul cheilor externe.
3. Să se creeze un declanşator care ar interzice ştergerea modelelor produselor producătorului *‚A'*, produselor care costă mai mult de 300, precum şi editarea valorilor câmpului ***Cod***. Se va afişa un mesaj de avertizare în cazul tentativei de a încălca constrângerea.
4. Să se creeze un declanşator DDL care ar interzice modificarea coloanei ***Model*** în tabelele bazei de date ***calculatoare*** cu afişarea mesajului respectiv.
5. Să se creeze un declanşator DDL care ar interzice modificarea schemei bazei de date în afara orelor de lucru.
6. Să se creeze un declanşator DDL care, la modificarea proprietăţilor coloanei ***Pret*** dintr-un tabel, ar face schimbări asemănătoare în mod automat în restul tabelelor.

10. RECUPERAREA BAZEI DE DATE

În capitolul de față sunt prezentate problematicile cu privire la importanța și utilitatea practică a copiilor de rezervă, regulile de creare a copiilor și formele de recuperare a datelor, principalele tipuri de dispozitive necesare pentru copiile de rezervă și diferite metode de realizare a copiilor de rezervă și de recuperare a datelor, precum și vizualizarea proprietăților copiilor de rezervă [4].

Microsoft SQL Server permite crearea copiilor de rezervă și recuperarea bazei de date. Componentele de recuperare și restaurare oferă o protecție serioasă a datelor stocate în bazele de date SQL Server. O strategie bine planificată de creare a copiilor de rezervă și restaurare asigură protejarea bazelor de date împotriva pierderilor de date cauzate de o serie de eșecuri.

O copie a datelor, care poate fi folosită pentru recuperarea și restabilirea datelor, se numește copie de rezervă sau *backup*. Prin utilizarea backupurilor de calitate poate fi asigurată protecția datelor după o serie de eșecuri, cum ar fi:

- Erori ale utilizatorilor, de exemplu, ștergerea din greșeală a unui tabel.
- Erori provenite din cauza hardware-ului, de exemplu, deteriorarea unei unități de disc sau pierderea completă a unui server.
- Dezastre naturale.

În plus, copiile de rezervă ale unei baze de date pot fi utile în scopuri administrative de rutină, cum ar fi copierea bazei de date de pe un server pe altul, crearea unei oglinzi a bazei de date sau arhivarea.

Înainte de a trece la analiza detaliată a procesului de creare a copiilor de rezervă, se vor examina trei modele posibile de recuperare a bazei de date.

10.1. Modele de recuperare și tipuri de backupuri

10.1.1. Modele de recuperare

Operațiile de salvgardare și restaurare în sistemul SQL Server intervin în contextul modelului de recuperare a bazei de date. Modelele de recuperare sunt concepute pentru a controla mentenanța jurnalului de tranzacții. Un model de recuperare este o proprietate a bazei de date care controlează modul în care tranzacțiile sunt jurnalizate, specifică dacă jurnalul de tranzacții necesită (și autorizează) crearea copiilor de rezervă și precizează tipurile de operații de restaurare disponibile.

Există trei modele de recuperare: simplă, completă și masivă, folosind jurnalul de tranzacții (*Bulk logged*). În general, o bază de date utilizează modelul de recuperare complet sau în modelul de recuperare simplu. Este posibil să se modifice modelul de recuperare a unei baze de date în orice moment.

Modelul de recuperare simplă

Nu utilizează nici un jurnal al copiilor de rezervă. Recuperarea datelor este posibilă numai din cea mai recentă copie de rezervă a bazei de date. Nu este posibilă recuperarea la o anumită dată și oră

Modelul de recuperare completă

Pentru restabilirea datelor necesită executarea periodică a procesului de salvgardare, folosind jurnalul de tranzacții și a backup-ului complet al bazei de date. În funcție de tipul de cădere a sistemului, baza de date poate fi recuperată, fie în punctul ultimului backup complet, fie în punctul ultimului backup, folosind jurnalul de tranzacții. În cazul în care jurnalul de tranzacții este încă disponibil, pot fi recuperate datele celor mai recente tranzacții validate. Modelul de recuperare completă face posibilă restabilirea bazei de date la o anumită dată și oră.

Modelul de recuperare masivă, folosind jurnalul de tranzacții

Modelul de recuperare ***bulk logged*** reprezintă un mod special de recuperare, care minimizează activitatea jurnalului de tranzacții în timpului operațiilor bulk. Nu este posibilă recuperarea la o anumită dată și oră.

Modelul de recuperare poate fi modificat în cadrul instrucțiunii *ALTER DATABASE* sau in fereastra ***Database Properties*** din consola de management a SQL Server.

10.1.2. Backupuri complete ale bazei de baze

O copie de rezervă poate avea ca scop o bază de date întreagă, o parte a bazei de date sau un set de fișiere sau de grupuri de fișiere. Pentru fiecare din aceste opțiuni, SQL Server susține backupuri complete și diferențiale.

Un backup complet al bazei de baze (***Full database backup***) include toate obiectele bazei de date, tabelele de sistem și datele. Tranzacțiile care au loc în timpul salvgardării sunt, de asemenea, înregistrate. Backupurile complete ale bazei de date permit o restabilire completă a bazei de date, care a existat la momentul efectuării operației de salvgardare.

10.1.3. Backupuri diferențiale

Un backup diferențial (***Differential backup***) se bazează pe cea mai recentă copie de rezervă completă a datelor. Acest lucru este cunoscut sub numele de *bază diferențială* sau *bază de diferențe*. Un backup diferențial conține doar datele care au fost schimbate după crearea backupului precedent. De obicei, backupul diferențial se creează mai repede decât un backup complet. Prin urmare, folosind backupul diferențial pot fi accelerate procesele de creare a backupurilor frecvente, cu scopul de a reduce riscul de pierdere a datelor. De obicei, bază diferențială este utilizată de mai multe backupuri succesive diferențiale. În timpul restabilirii datelor, în primul rând, este restabilit backupul complet, urmat de cel mai recent backup diferențial.

Pe parcursul când baza de date este actualizată, volumul de date care sunt incluse în backupurile diferențiale crește. Acest fapt face ca procedura de creare și recuperare să decurgă mai lent. În cele din urmă, un alt backup complet va trebui să fie creat pentru a oferi o nouă bază completă pentru alte serii de backupuri diferențiale.

Fiecare copie de rezervă a datelor include o parte din jurnalul de tranzacții, astfel încât să fie posibilă recuperarea datelor până la momentul creării acestei copii.

10.1.4. Backupuri ale bazei de date

Backupul bazei de date (***Database backup***) este ușor de folosit și este recomandat ori de câte ori dimensiunea bazei de date permite acest lucru. SQL Server suportă următoarele tipuri de backupuri ale bazei de date:

- ***Full Database backup.*** O copie de rezervă a întregii baze de date, care reprezintă toată baza de date la momentul în care procesul de creare a backupului s-a terminat.
- ***Differential database backups.*** O copie de rezervă a tuturor fișierelor din baza de date. Acest backup conține numai acele extenduri de date care au fost modificate din momentul ultimei copii a fiecărui fișier.

10.1.5. Backupuri parțiale

Backupurile parțiale și parțiale diferențiale au fost introduse, pentru prima dată, în SQL Server 2005. Aceste backupuri sunt concepute pentru a asigura o flexibilitate mai mare sub un model simplu de recuperare pentru susținerea bazelor de date care conțin unele grupuri de fișiere de tip *read-only*. Cu toate acestea, acest tip de backup este suportat de toate modelele de recuperare.

SQL Server 2014 susține următoarele tipuri de backupuri parțiale ale fișierelor:

- ***Backup parțial.*** O copie de rezervă a tuturor datelor din grupul de fișiere primare, din orice grup de fișiere de citire/scriere, precum și din orice fișier sau grup de fișiere *read-only* specificate opțional. Un backup parțial al unei bazei de date *read-only* conține numai grupul de fișiere primare.
- ***Backup parțial diferențial.*** O copie de rezervă, care conține doar extenduri de date, ce au fost modificate începând cu cel mai recent backup parțial al aceluiași set de grupuri de fișiere.

10.1.6. Backupuri de fișiere

Într-o bază de date, fișierele pot fi supuse unui proces de creare a copiilor și restabilire individual. Folosind *backupul de fișiere,* poate fi mărită viteza de recuperare, restabilind numai fișierele deteriorate, fără restabilirea restului bazei de date. De exemplu, în cazul în care o bază de date este formată numai din câteva fișiere care sunt amplasate pe diferite discuri și un disc s-a deteriorat, doar fișierul de pe discul deteriorat se cere, de fapt, să fie recuperat. Cu toate acestea, planificarea și restaurarea *backupului de fișiere* poate fi complexă și, de aceea, *backupul de fișiere* trebuie să fie utilizat doar în cazul în care el este univoc necesar în planul de restabilire.

SQL Server admite următoarele tipuri de *backup de fișiere*:

- ***Backup de fișiere.*** O copie de rezervă completă a tuturor datelor unui sau mai multor fișiere, sau grupuri de fișiere.

  În cadrul unui model de recuperare simplă, backupul de fișiere poate fi făcut asupra grupurilor de fișiere secundare *read-only,* dar și asupra fișierelor de *citire/scriere*. În afară de aceasta, fișierele de *citire/scriere* trebuie copiate toate împreună. Acest lucru va asigura ca baza de date să fie repusă într-o stare consistentă.
- ***Backup diferențial de fișiere.*** O copie de rezervă a unui sau mai multor fișiere care conțin numai extenduri care s-au modificat de la cel mai recent backup complet al fiecărui fișier.

10.1.7. Backupuri ale jurnalului de tranzacții (modelele complete și Bulk-Logged)

Modelele de recuperare completă sau bulk-logged, presupun executarea sistematică a backupurilor jurnalelor de tranzacții (sau backupul jurnalelor). Fiecare backup al jurnalului se referă la o parte din jurnalul de tranzacții, care a fost activ în momentul creării backupului, și include toate înregistrările de jurnal care nu au fost salvate în backupul precedent. O succesiune neîntreruptă de backupuri ale jurnalelor conține lanțul complet de jurnale ale bazei de date, care se consideră neîntrerupt. În modelul de recuperare completă, iar uneori și în modelul de recuperare Bulk-Logged, un lanț neîntrerupt de jurnale permite restaurarea bazei de date până la orice moment de timp.

Înainte de crearea primului jurnal de rezervă, mai întâi, trebuie creat un backup complet al bazei de date. Apoi, sistematic, se fac backupuri ale jurnalului de tranzacții nu numai pentru a minimiza pierderile rezultatelor muncii, dar și pentru a trunchia jurnalul de tranzacții.

10.1.8. Backupuri copy-only

Backupul copy-only (*Copy-only backup*) reprezintă o copie de rezervă independentă de mecanismele de salvgardare şi de secvenţa de backupuri convenţionale din sistemul SQL Server. În mod normal, o copie de rezervă modifică baza de date şi afectează restaurarea copiilor ulterioare. Uneori, este util să se facă o copie pentru un anumit scop, fără a afecta procedurile globale de salvgardare şi restaurare a bazei de date. Pentru aceasta, se poate folosi o copie de rezervă *copy-only*.

Tipurile de backupuri *copy-only* sunt după cum urmează:

- ***Copy-only full backups (toate modelele de recuperare).*** Copia de rezervă copy-only nu poate servi nici bază diferenţială, nici în calitate de backup diferenţial şi nu are nici un efect asupra bazei diferenţiale. Restaurarea unei copii de rezervă copy-only este identică cu restabilirea oricărui altui backup complet.
- ***Copy-only log backups (numai modelele de recuperare completă şi masivă, folosind jurnalul de tranzacţii).*** Copia de rezervă *copy-only log* păstrează punctul de arhivă al jurnalului existent şi, prin urmare, nu afectează ordinea de backupuri log regulare. *Backupurile log* de tipul *copy-only* sunt de obicei inutile. Cu toate acestea, există posibilitatea să se creeze o rutină nouă de log backup (folosind *WITH NORECOVERY*) şi să fie folosită această copie de rezervă cu orice alte copii anterioare necesare pentru a restabili ordinea. Cu toate acestea, *Backupul copy-only log* poate fi uneori util doar pentru a efectua o restaurare on-line.

10.2. Restricţii asupra operaţiilor de recuperare

SQL Server permite crearea copiilor de rezervă în timpul utilizării bazei de date. Cu toate acestea, există unele restricţii.

10.2.1. Nu pot fi create copii ale datelor deconectate (offline)

Orice operaţie de creare a copiilor de rezervă, care implicit sau explicit, se referă la date deconectate, va eşua. Unele exemple tipice sunt prezentate în continuare:

- Fie că se doreşte o copie de rezervă completă a bazei de date, dar un grup de fişiere din baza de date este deconectat. Deoarece într-un backup complet sunt incluse implicit toate grupurile de fişiere ale bazei de date, realizarea acestei sarcini este imposibilă.

  În acest caz, poate fi solicitat un backup de fişiere pentru care să fie specificate doar fişierele ce sunt în regim *online*.
- În cazul în care se cere un backup parţial, dar grupul de fişiere de citire/scriere este deconectat, crearea copiei de rezervă va fi, de asemenea, imposibilă, deoarece pentru un backup parţial sunt necesare toate grupurile de fişiere citire/scriere.
- Fie că se cere un backup de fişiere specificate, dar unul din fişiere nu este în regim operativ (*online*). Operaţia de creare a copiei de rezervă nu va reuşi. Pentru a efectua un backup al fişierelor, mai întâi din lista de fişiere trebuie şterse fişierele *offline* şi apoi realizată operaţia.

De obicei, un backup al jurnalului de tranzacţii are loc chiar dacă unul sau mai multe fişiere de date nu sunt disponibile. Cu toate acestea, pentru executarea cu succes a backupului, în cazul în care cel puţin un fişier conţine modificări ***bulk-logged*** (masive) petrecute în modelul de recuperare ***bulk-logged***, toate fişierele trebuie să fie online.

10.2.2. Restricţii asupra accesului concurenţial în timpul creării copiei de rezervă

SQL Server utilizează un backup online, pentru a admite crearea backupului bazei de date în timp ce baza de date este folosită. Astfel, în timpul backupului, majoritatea operaţiilor rămân posibile. De exemplu, instrucţiunile *INSERT*, *UPDATE* sau *DELETE* sunt acceptate în timpul operaţiei de creare a copiilor de rezervă. Cu toate acestea, dacă se încearcă lansarea unui backup în timp ce se creează sau se şterge un fişier al bazei de date, operaţia de copiere va fi amânată până la terminarea operaţiei de creare sau de ştergere a fişierului sau timpul când trebuie formată copia de rezervă poate fi depăşit.

Următoarele operații nu pot fi executate în timpul unui backup al bazei de date sau unui backup al jurnalului de tranzacții:

- Operațiile de gestionare a fișierelor, cum ar fi instrucțiunile *ALTER DATABASE* cu opțiunile *ADD FILE* sau *REMOVE FILE*.
- Operațiile de comprimare a bazei de date sau de comprimare a unui fișier. Aici se includ și operațiile de autocomprimare.
- Operațiile de creare sau ștergere a unui fișier din baza de date în timp ce operația de backup este în curs. Operația de creare sau de ștergere eșuează.

În cazul în care operația de creare a copiei de rezervă se suprapune cu o operație de gestionare a fișierelor sau o operație de comprimare, apare un conflict. Indiferent de faptul care din operațiile ce conflictualizează a fost lansată prima, cea de-a doua operație așteaptă expirarea timpului de blocare stabilit de prima operație (perioada pauzei este controlată de setarea sesiunii time-out.). În cazul în care deblocarea are loc până la expirarea timpului perioadei de pauză (time-out), cea de a doua operație continuă. În caz contrar, a doua operație eșuează.

Este cunoscut faptul că un backup în baza de date este o operație de copiere a datelor pe un dispozitiv fizic. Acest lucru poate fi realizat, folosind ***SQL Server Management Studio*** sau instrucțiunea *BACKUP*.

Precum s-a reiterat, pentru a face un backup, nu se cere stoparea sistemului sau deconectarea de la sistem a utilizatorilor acestuia. Copierea se realizează cu utilizatorii conectați, dar, totuși, trebuie menționat că, în acest timp, în unele cazuri poate scădea performanța sistemului. Din acest motiv, se alege ora cu cea mai mică activitate a serverului.

10.3. Crearea copiilor de rezervă

10.3.1. Dispozitive backup

Un dispozitiv backup reprezintă un disc dur sau alt dispozitiv de depozitare a datelor care conține unul sau mai multe fișiere backup. Un fișier backup este un fișier obișnuit al sistemului de operare. Dimensiunea maximală a fișierului backup este determinată de spațiul disponibil pe dispozitiv. Prin urmare, dimensiunea potrivită pentru un dispozitiv backup depinde de dimensiunea fișierelor backup.

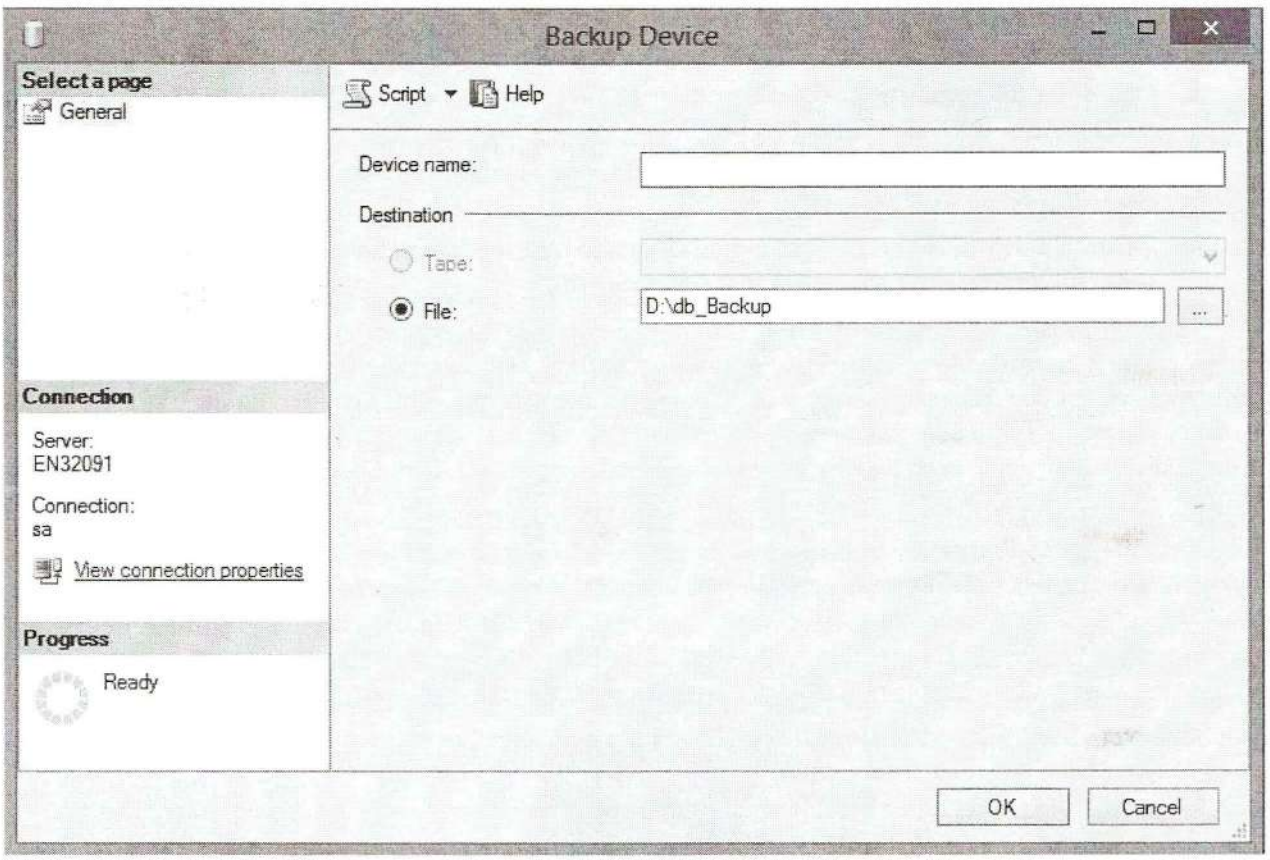
Se recomandă ca fișierele backup să fie păstrate pe dispozitive aparte de baza de date sau fișierele log. Acest lucru este necesar pentru a garanta accesul la fișierele backup în cazul defectării discurilor cu date sau loguri (jurnale de tranzacții).

Un dispozitiv backup pe interfața Management Studio nu este altceva decât un pointer spre locul în care backupul bazei de date va fi plasat. În ***SQL Server Management Studio*** poate fi creat un dispozitiv backup al oricărei instanțe, sau se poate crea când, de facto, se face un backup al bazei de date. Un dispozitiv backup poate specifica numele unui fișier pe discul rigid care va fi scris când backupul va fi executat, sau poate fi specificată o unitate externă de bandă.

Pentru definirea unui dispozitiv backup în ***SQL Server Management Studio***, se procedează în modul următor:

1. În ***Object explorer*** se extinde serverul care va recepționa backupul.
2. Se extinde dosarul ***Server Objects***
3. Se face un click cu butonul drept al mouse-ului pe dosarul ***Backup Devices***
4. În meniul de context, care s-a deschis, se selectează opțiunea ***New Backup Device***

   În acest punct, va fi deschisă fereastra ***Backup Device***:



5. În câmpul de editare ***Device Name***, se introduce numele dispozitivului backup care va fi utilizat de instrucțiunea de creare a copiei de rezervă. În cazul nostru, se alege numele ***device01***.
6. Se aplică activități în grupul ***Destinations***:

   În câmpul ***Tape*** se alege dintr-o lista un dispozitiv de destinație de tipul ***Tape*** (bandă magnetică). Această opțiune este disponibilă numai dacă biblioteca de benzi magnetice este atașată la calculatorul unde rulează motorul bazelor de date Microsoft SQL Server.

   În câmpul de editare ***File***, se indică locul fizic al fișierului pentru un dispozitiv de backup existent, sau se specifică fișierul-destinație pentru un dispozitiv backup logic nou.

   Pentru un dispozitiv de backup existent, sistemul afișează calea către fișierul backup. Câmpul ***File*** nu este editabil, iar butonul **Browse** nu este disponibil.

   Calea către dispozitivul de tip ***File*** este citită, implicit, din setările serverului SQL în cauză și anume din setările ***Default Database Locations,*** optiunea ***Backup***..

   Pentru un nou dispozitiv backup logic, trebuie specificată calea fișierului backup pentru care este definit acest dispozitiv. Acest fișier nu neapărat trebuie deja să existe.

   Pentru a specifica un fișier backup local, poate fi utilizat butonul **Browse** ce se află în partea dreaptă a căsuței de inserție ***File***. Prin intermediul casetei de dialog ***Locate Database Files,*** se navighează prin spațiul accesibil de fișiere. Dacă fișierul de backup încă nu există, numele lui poate fi specificat în câmpul ***File name*** al acestei casete de dialog.

   Câmpul ***File*** poate fi editat și în mod manual, suprascriind calea implicită, numele fișierului și extensia.

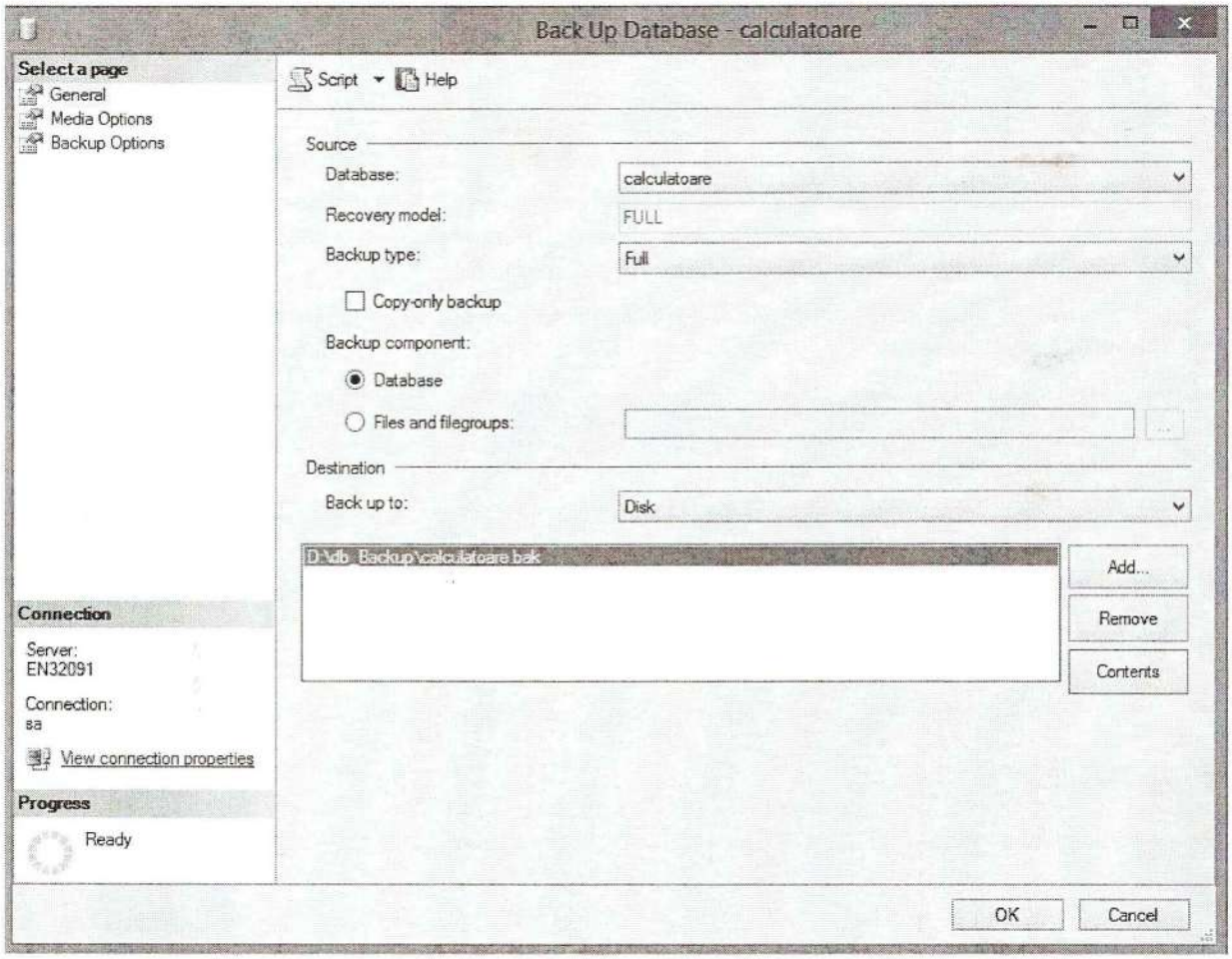
   După crearea dispozitivului, acesta se va găsi în ***Object Explorer***, grupul ***Backup Defices***.

10.3.2. Realizarea unui backup complet al bazei de date

Pentru realizarea unui backup complet al bazei de date, se parcurg următorii paşi:

1. În ***SQL Server Management Studio***, se selectează baza de date pentru care se va realiza un backup.
2. Se face un click cu butonul drept al mouse-ului pe baza de date şi, în meniul de context, ce s-a deschis, se aplică un click pe opţiunea ***Tasks***, apoi se trece şi se acţionează opţiunea ***Back Up...***

   În acest punct, se va deschide fereastra ***Back Up Database*** propriu-zisă:



3. În meniul derulant ***Database***, se selectează baza de date care va fi copiată.
4. În ***Recovery model***, poate fi văzut modelul folosit pentru restabilire (*SIMPLE*, *FULL* sau *BULK_LOGGED*) afişat pentru baza de date selectată.
5. În lista de selecţie ***Backup type***, se selectează tipul de backup (***Full***, ***Differential*** sau ***Transaction log***) care va fi executat asupra bazei de date specificate. În cazul nostru, este aleasă opţiunea ***Full*** (complet).
6. În grupul ***Backup component***, se selectează componentele bazei de date pentru care vor fi create copii de rezervă. Dacă în ***Backup type*** este selectată opţiunea ***Transaction Log***, această opţiune nu este accesibilă. Poate fi selectată una din următoarele 2 opţiuni: ***Database***, ***Files and filegroups***.
7. În ***Destination***, se selectează dispozitivul şi tipul acestuia - disc sau bandă. Pentru selectarea dispozitivului, se face un click pe butonul **Add**. În acest punct, va apărea o casetă de dialog care permite căutarea dispozitivelor backup logice şi fizice existente.

8. În această casetă de dialog, se face un click pe opțiunea ***Backup device*** și se selectează unul din dispozitivele create pentru backup sau se alege numele fișierului de pe discul rigid, și apoi se aplică un click pe butonul **OK**. După această acțiune, controlul procesului se întoarce ferestrei ***Back Up Database***.

   În fereastra ***Back Up Database***, butoanele **Remove** și **Contents** au următoarele semnificații:

   - **Remove** suprimă unul sau mai multe dispozitive din listă.
   - **Contents** afișează metainformația despre dispozitivul selectat.

9. După configurarea respectivă, e suficient să se aplice un click pe butonul **OK** și, în acest punct, va apărea o fereastră cu un mesaj despre crearea cu succes a copiei de rezervă.

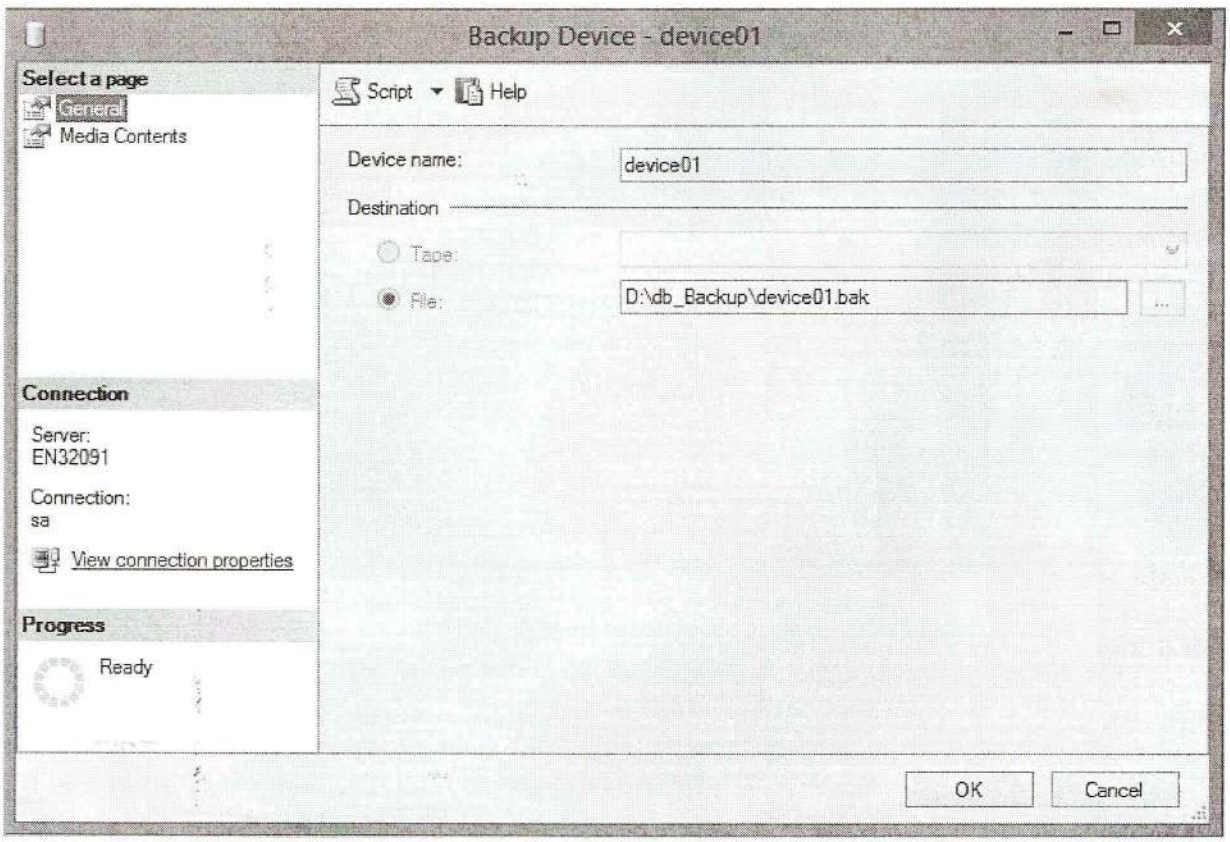
10. În această fereastră se aplică un click pe butonul **OK** și procesul este finalizat.

10.3.3. Vizualizarea copiei de rezervă

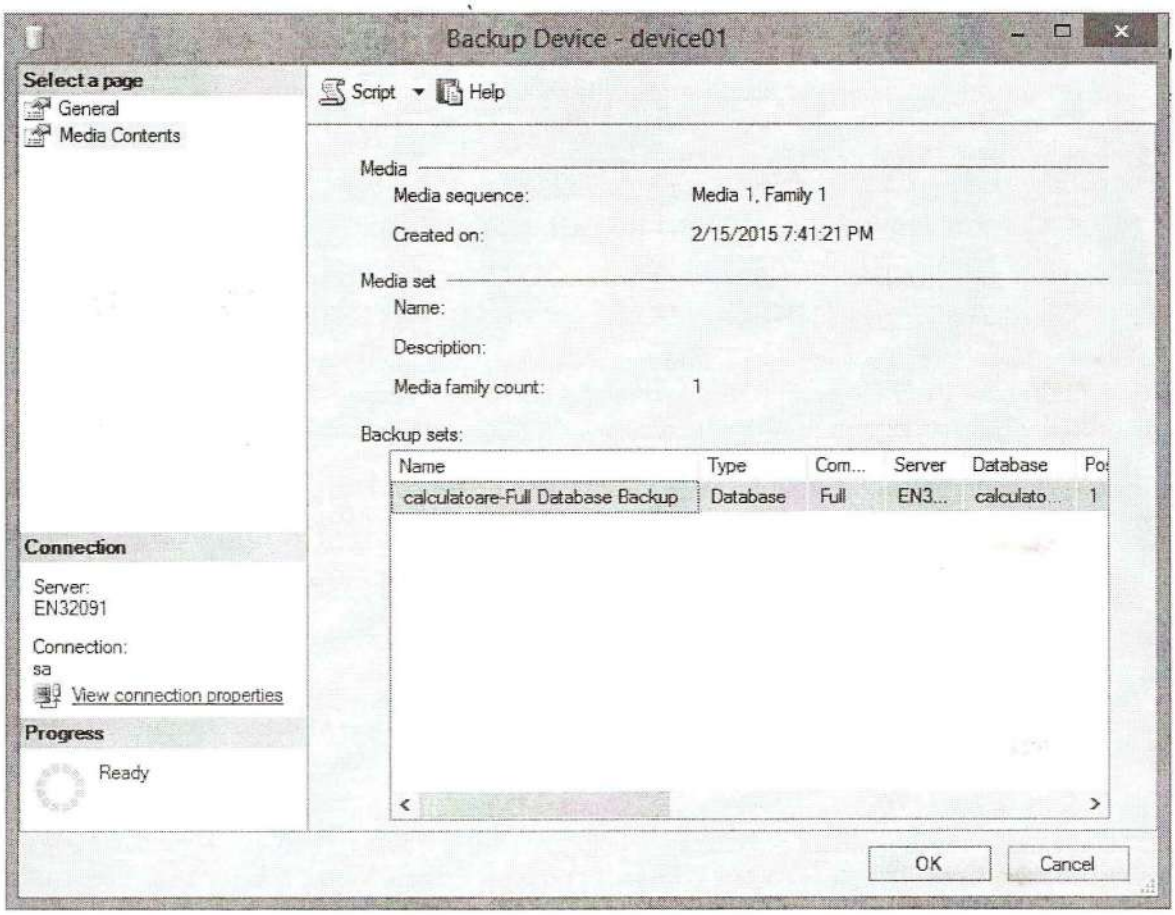
Realizarea backupului poate fi verificată. Pentru aceasta, se parcurg următorii pași:

1. În ***Management Studio***, se extinde serverul respectiv, apoi se extinde dosarul ***Server Objects*** și se face un click pe dosarul ***Backup Devices***. Se poate observa că, în acest punct, în partea dreaptă a ferestrei, se va găsi dispozitivul backup creat.

   Acum, se aplică un dublu-click asupra dispozitivului backup. În acest punct, va apărea următoarea fereastră:



2. În fereastra ***Backup device***, sunt prezentate dispozitivul și calea acestuia, precum se vede în câmpul ***File.*** Pentru a vedea fișierele cu copiile propriu-zise, e suficient să se aplice un click pe pagina ***Media Content*** din partea stângă a ferestrei:



Această fereastră conține informații despre suportul de date individual, setul de suporturi de date şi seturile de backupuri.

Media

Un disc sau un set de benzi magnetice pe care se stochează copia de rezervă.

Media Sequence

Afişează numărul secvențial al suportului de date, numărul secvențial al grupului, identificatorul oglinzii. Fiecare suport backup este dotat cu un număr secvențial care indică ordinea în care acest dispozitiv este utilizat.

Media Set

Un set de suporturi de date este o colecție ordonată de suporturi pentru care una sau mai multe operații backup au fost scrise prin utilizarea unui număr constant de dispozitive backup.

Name

Afişează numele setului de suporturi de date, dacă există.

Description

Afişează descrierea setului de suporturi de date, dacă există.

Media family count

Afişează numărul de familii într-un set de suporturi de date. Fiecare set de suporturi de date este o colecție de una sau mai multe familii de suporturi de date. Toate datele de ieşire la un anumit dispozitiv backup (sau grup de dispozitive backup oglindă) formează o singură familie de suporturi de date. Fiecare set de suport de date conține o familie de suporturi separate pentru fiecare dispozitiv (sau grup de dispozitive oglindă). De exemplu, în cazul în care un set de suporturi de date foloseşte două dispozitive backup neoglindă, atunci setul de suporturi de date conține două familii de suporturi.

Backup sets

Afişează informaţia despre setul sau seturile de backupuri ce se conţin pe suportul de date. Un set backup este rezultatul unei operaţii backup reuşite, al cărui conţinut este distribuit între suporturile din setul de dispozitive backup.

10.3.4. Crearea backupului în Transact-SQL Query Editor

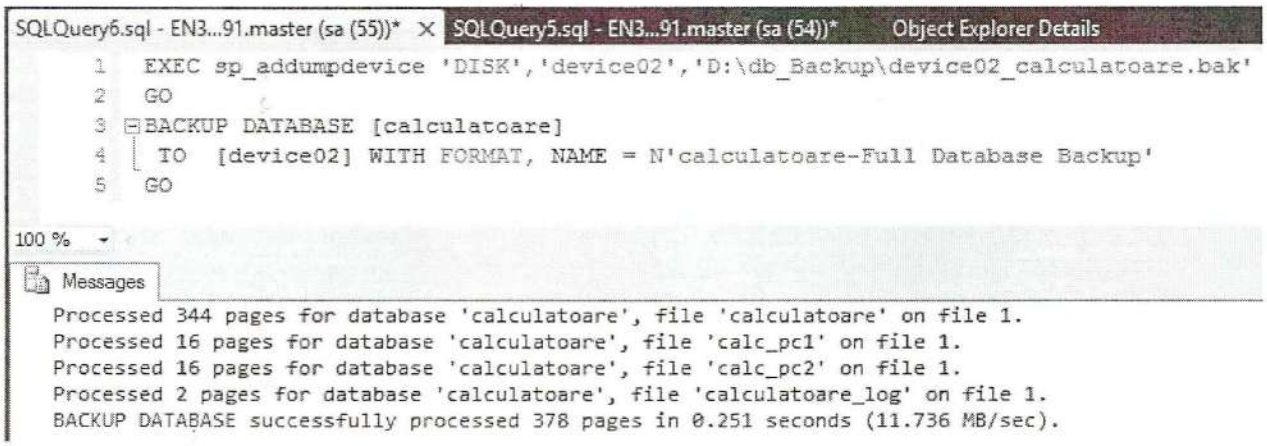
Construirea unei copii de rezervă poate fi executată şi în mediul ***Transact-SQL Query Editor***. Mai jos, sunt prezentate câteva variante de realizare a unei copii de rezervă a bazei de date ***calculatoare***.

Pentru crearea unui backup, mai întâi, se recomandă să se definească un dispozitiv backup cu proprietăţi dezirabile. Pentru definirea dispozitivului backup în ***Transact-SQL Query Editor***, poate fi utilizată procedura stocată de sistem ***sp_addumpdevice*** care are următoarea sintaxă de utilizare:

```
sp_addumpdevice [ @devtype = ] 'tip_dispozitiv'
        , [ @logicalname = ] 'nume_logic'
        , [ @physicalname = ] 'nume_fizic'
      [ , { [ @cntrltype = ] tip_controller |
          [ @devstatus = ] 'statut_dispozitiv'}
      ]
```

În exemplul ce urmează, prima instrucţiune creează dispozitivul backup nou. În calitate de parametri de intrare este definit tipul dispozitivului – *DISK*, numele logic al dispozitivului – ***device02*** şi numele fizic al dispozitivului, specificând în acelaşi timp şi cale spre acesta.

Instrucţiunea *BACKUP* ce urmează creează un backup complet al bazei de date ***calculatoare*** în dispozitivul de backup creat anterior. Clauza *FORMAT* se utilizează atunci când, prima dată, se face copia de rezervă sau când se doreşte să se rescrie o copie de rezervă. Clauza *NAME* specifică numele setului de backupuri.



În exemplele de mai jos, se definesc două dispozitive backup logice noi ***backup_Files*** şi ***backup_Log***. Instrucţiunea *BACKUP LOG* creează o copie de rezervă a jurnalului de tranzacţii al bazei de date ***calculatoare***. Instrucţiunea *BACKUP DATABASE* creează un backup pentru fişierele ce fac parte din grupuri de fişiere secundare ale bazei de date calculatoare ***fgrup1*** şi ***fgrup2***. Fişierele copiate în ambele cazuri de utilizare a instrucţiunii *BACKUP* sunt salvate pe dispozitivele backup respective.

```
USE master
GO
EXEC sp_addumpdevice 'DISK', 'backup_Files',
                'D:\db_Backup\calculatoare_Files.bak';
GO
EXEC sp_addumpdevice 'DISK', 'backup_Log',
                'D:\db_Backup\calculatoare_Log.bak';
```

```
GO
BACKUP DATABASE [calculatoare]
       FILEGROUP = N'fgrup1',
       FILEGROUP = N'fgrup2'
       TO backup_Files
GO
BACKUP LOG calculatoare TO backup_Log
GO
```

Instrucțiunea *BACKUP* poate conține și alte elemente, în funcție de scopurile urmărite.

Urmează un exemplu care afișează informații despre copia de rezervă:

```
RESTORE FILELISTONLY FROM device01;
```

Această instrucțiune afișează din tabelul ***backupfile*** al bazei de date ***msdb*** informații despre lista fișierelor și jurnalul de tranzacții de pe dispozitivul backup ***device01***.

10.4. Restaurarea backupului

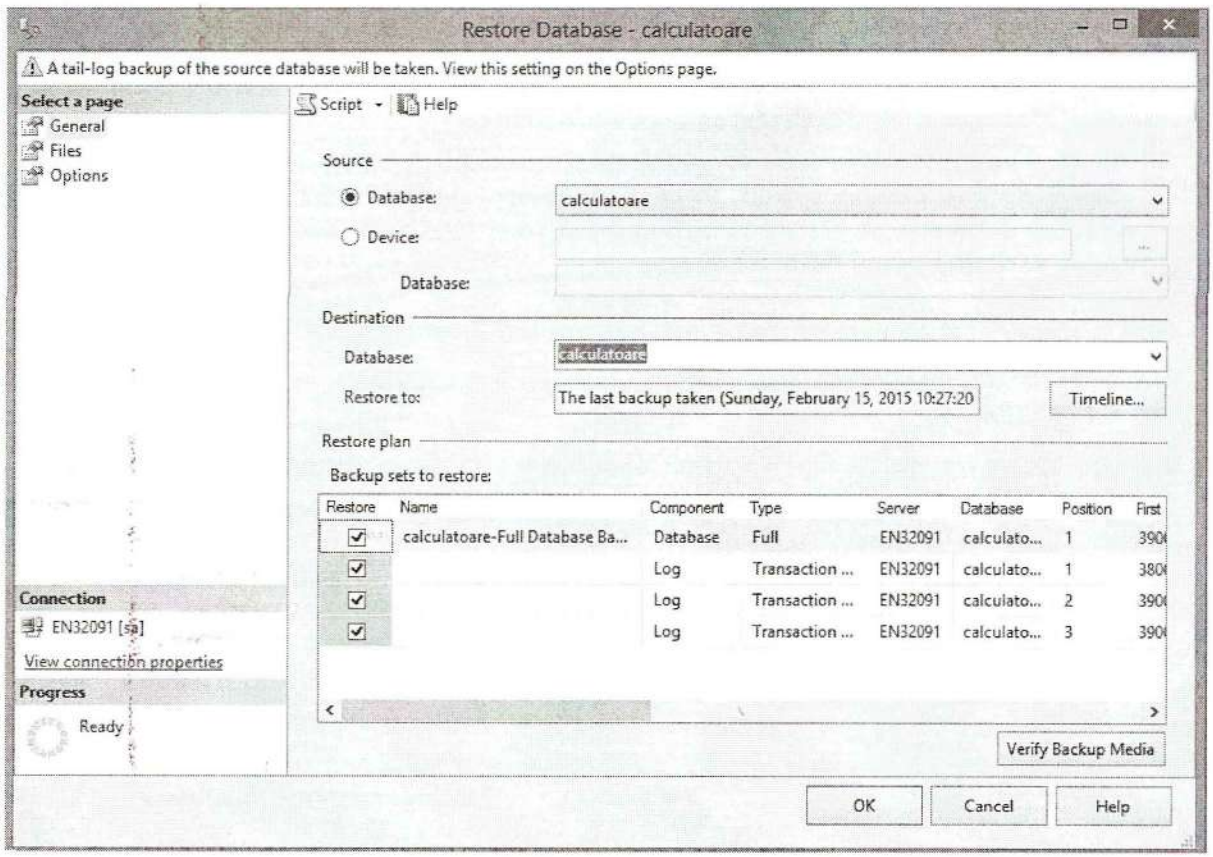
Procesul de restaurare este foarte asemănător procesului de copiere.

10.4.1. Recuperarea datelor în mediul SQL Server Management Studio

Pentru recuperarea datelor în mediul ***SQL Server Management Studio*** trebuie desfășurate următoarele activități:

1. Se face un click cu butonul drept al mouse-ului asupra bazei de date, care va suporta restaurarea unei copii și, în meniul de context, care se va deschide, se vor selecta opțiunea ***Tasks*** și apoi opțiunea ***Restore,*** apoi ***Database...***sau ***Files and Filegroups...***, în funcție de tipul ales de restaurare

   În acest punct, va fi deschisă următoarea fereastră:



În fereastra ***Restore database***, pagina ***General***, sunt disponibile două secțiuni de parametri: ***Source*** și ***Destination.***

2. Secțiunea ***Source*** se utilizează pentru specificarea sursei și locației copiilor de rezervă ce trebuie restabilite. Să se selecteze una din următoarele opțiuni:
   - ***Database.*** Din lista derulantă se selectează baza de date ce trebuie recuperată. Lista conține numai acele baze de date, care au fost introduse în istoricul ***msdb*** copiilor de rezervă.

     Atunci când copia de rezervă este luată de pe alt server, serverul destinație nu va dispune de istoricul acesteia pentru baza de date specificată. In acest caz, pentru a specifica manual fișierul sau dispozitivul pentru restabilire, se selectează opțiunea ***Device***.
   - ***Device.*** Faceți click pe butonul **Browse** (...) pentru a deschide caseta de dialog ***Select backup devices***. În caseta ***Backup media type*** selectați unul din tipurile de dispozitive disponibile. Pentru a selecta unul sau mai multe dispozitive backup, faceți un click pe butonul **Add**.

     După adăugarea unităților dorite din lista ***Backup media***, faceți click pe **OK** pentru a reveni la pagina ***General***.

     Tot aici, în secțiunea ***Source,*** opțiunea ***Device,*** lista ***Database*** se selectează numele bazei de date ce trebuie restabilită.
3. În secțiunea ***Destination:***
   - Câmpul ***Database*** este automat populat cu numele bazei de date care urmează să fie restabilită. Pentru a schimba numele bazei de date, se introduce numele nou în caseta respectivă.
   - În câmpul ***Restore to***, pentru a lua ultimul backup, se lasă valoarea implicită sau, pentru a selecta manual o limită specifică pentru a opri acțiunea de recuperare a bazei de date, se face clic pe butonul ***Timeline***.
4. În partea de jos a ferestrei se afla un grid, ***Backup Sets to Restore,*** unde se selectează backupurile pentru restabilire. Această grilă afișează copiile de rezervă disponibile pentru locația specificată. În mod implicit, sistemul sugerează un plan de restabilire. Pentru a înlocui planul de restabilire sugerat, utilizatorul are posibilitatea să debifeze obiectele selectate din grila. Backupurile care depind de restabilirea unei copii de rezervă anterioare sunt automat deselectate atunci când backupul anterior este deselectat.

   În cazul nostru, se alege sursa ***From Device*** și se apasă butonul **Browse**. În fereastra de dialog ***Select Backup Devices*** apărută, se alege, în câmpul derulant ***Backup Media type,*** opțiunea ***Backup Device*** și se găsește numele dispozitivului backup necesar, apăsând butonul **Add**. Apoi, se apasă butonul **OK.**

10.4.2. Recuperarea datelor, folosind instrucțiuni Transact-SQL

Pentru restaurarea bazelor de date, folosind limbajul Transact-SQL, se utilizează instrucțiunea *RESTORE DATABASE.*

În continuare, se vor analiza câteva exemple de utilizare a acestei instrucțiuni.

Exemplul de mai jos realizează restaurarea bazei de date ***calculatoare*** din dispozitivul logic ***device01***:

```
RESTORE DATABASE calculatoare FROM device01 WITH REPLACE;
```

Opțiunea *WITH REPLACE,* în instrucțiunea de mai sus, indică faptul că conținutul curent al bazei de date va fi suprascris de cel din backup.

Următorul exemplu execută restabilirea backupului complet al bazei de date ***calculatoare*** după care urmează o operație de restabilire jurnalului de tranzacții. Backupul jurnalului de tranzacții face parte din backupul complet al bazei de date. Prin aceasta se explică folosirea aceluiași dispozitiv backup.

```
RESTORE DATABASE calculatoare FROM device02 WITH NORECOVERY, REPLACE;
RESTORE LOG calculatoare
   FROM DISK = 'D:\db_Backup\device02_calculatoare.bak'
   WITH RECOVERY;
```

Precum se vede, restabilirea poate fi derulată, indicând numele dispozitivului de backup sau indicând calea explicită către un fișier ****.bak.***

Opțiunea *NORECOVERY* specifică faptul că *roll back*-ul nu are loc. Acest lucru permite operației *roll forward* să continue cu următoarea instrucțiune din secvență. În acest caz, se poate restaura succesiunea de backupuri și rularea lor înainte.

Opțiunea *RECOVERY* (este implicită pentru sistem) indică faptul că operația *roll back* ar trebui să fie efectuată după ce roll forward-ul este terminat pe backupul curent.

Recuperarea bazei de date presupune că întregul set de date fiind restaurat (setul roll forward) este consistent cu baza de date. În cazul în care setul de roll forward-uri nu a fost rulat înainte pentru ca să fie destul de consistent cu baza de date și opțiunea *RECOVERY* e specificată, motorul bazei de date returnează o eroare.

Operațiile *roll back* și *roll forward* fac parte din planul general de restaurare a datelor în SQL Server. Dacă are loc restabilirea jurnalelor de tranzacții, atunci SQL Server transcrie toate datele despre tranzacțiile validate din jurnalul de tranzacții în baza de date. Această operație se numește *roll forward.* Apoi SQL Server detectează toate tranzacțiile nevalidate și le anulează (*roll back*).

Pentru realizarea cu succes a comenzilor menționate mai sus, instrucțiunea *RESTORE DATABASE* este definită cu opțiunea *NORECOVERY.* Trebuie menționat că backupul jurnalului de tranzacții trebuie să suporte recuperarea cu opțiunea *NORECOVERY.*

În instrucțiune de mai jos, se execută restabilirea unui singur fișier ***calc_pc1*** din grupul de fișiere ***fgrup1***. Fișierul restabilit este mutat într-un directoriu deosebit de cel implicit pentru baza de date calculatoare.

```
USE master
GO
RESTORE FILELISTONLY FROM device02;
RESTORE DATABASE calculatoare
  FILE = 'calc_pc1',
  FILEGROUP='fgrup1'
  FROM device02
  WITH MOVE 'calc_pc1' TO 'D:\dbdata\pc1.ndf';
GO
```

Instrucțiunea *RESTORE FILELISTONLY* determină numărul și denumirile fișierelor în backupul bazei de date ce permite realizarea acestei operații de restaurare.

Întrebări de control:

1. Modele de recuperare in SQL Server 2014.
2. Ce sunt backupurile și ce tipuri de backupuri susține SGBD-ul SQL Server 2014?
3. Care sunt regulile și constrângerile de realizare a backupurilor?
4. Care sunt dispozitivele backup? Metodele de creare a dispozitivelor backup în SQL Server. Proprietățile dispozitivelor backup.

5. Cum se restaurează un backup?
6. Care este structura instrucțiunilor SQL de creare şi restaurare a backupurilor?
7. Utilitatea opțiunilor *RECOVERY* şi *NORECOVERY*

Sarcini practice:

1. Să se creeze un dosar ***Backup_lab10***. Să se execute un backup complet al bazei de date calculatoare în acest dosar. Fişierul copiei de rezervă să se numească ***exercitiul1.bkp***. Să se scrie instrucțiunea SQL respectivă.
2. Să se scrie instrucțiunea unui backup diferențiat al bazei de date calculatoare. Fişierul copiei de rezervă să se numească ***exercitiul2.bkp***.
3. Să se scrie instrucțiunea unui backup al jurnalului de tranzacții al bazei de date calculatoare. Fişierul copiei de rezervă să se numească ***exercitiul3.bkp***
4. Să se execute restaurarea consecutivă a tuturor copiilor de rezervă create. Recuperarea trebuie să fie realizată într-o bază de date nouă ***calculatoare_lab10***. Fişierele bazei de date noi se află în dosarul ***BD _lab10***. Să se scrie instrucțiunile SQL respective

11. SERVICII DE INTEGRARE A DATELOR

Capitolul este consacrat în întregime utilizării serviciilor de integrare a datelor (***SQL Server Integration Services***). Sunt examinate utilitatea acestor servicii, metodele de transformare şi de export-import al datelor, şi gestiunea pachetelor acestor servicii în mediul ***SQL Server Management Studio*** [1,3,5].

SQL Server Integration Services (SSIS) este o platformă pentru construirea soluţiilor de integrare şi transformare a datelor. Utilizarea serviciilor de integrare poate rezolva o serie de probleme complexe de business prin automatizarea procesului de transfer al fluxurilor informaţionale între diferite structuri de date SQL Server şi sursele de date externe. Serviciul SSIS a substituit ***Data Transformation Services*** (*DTS*), care a apărut pentru prima dată în ***SQL Server 7.0.***

Începând cu versiunea SQL Server 2012, SSIS acceptă două modele de implementare (*deployment*): modelul de implementare a proiectului şi modelul de implementare a pachetului. Modelul de implementare a proiectului permite desfăşurarea proiectelor pe un server *Integration Services*.

În cadrul acestui capitol, va fi analizat modelul de implementare a pachetului, fiind unul de bază.

11.1. Componentele şi funcţionalităţile SSIS

Integration Services include: instrumente grafice şi asistenţi pentru construirea şi depănarea pachetelor de integrare a datelor; mecanisme de efectuare a operaţiilor ce folosesc FTP pentru executarea instrucţiunilor SQL sau pentru expedierea mesajelor e-mail; *API*-uri *(Application Programming Interfaces)* pentru utilizarea SSIS de către aplicaţiile externe.

Cu ajutorul serviciilor SSIS, date ale bazei de date SQL Server pot fi transformate în date compatibile reprezentate în diverse alte formate. Aceste servicii constituie o piesă fundamentală în procesul de colectare şi transformare a datelor.

Procesul de transformare permite importarea şi exportarea datelor, convertirea datelor între aplicaţiile care utilizează diverse formate ale datelor, astfel, încât aplicaţia locală sau externă are posibilitatea să lucreze cu datele celeilalte. Conversia presupune transformarea datelor, adică construirea unei secvenţe de operaţii ce pot fi executate asupra unei surse de date, înainte ca ele să fie scrise în punctul de destinaţie.

Arhitectura unui pachet SSIS este, în realitate, o descriere detaliată a operaţiilor care trebuie executate pentru ca o unitate de date să fie importată, exportată sau transformată.

Serviciile SSIS se pot conecta la o varietate largă de date-sursă care îmbină multiple surse într-un pachet. Un pachet se poate conecta la bazele de date relaţionale, folosind tehnologiile .NET şi OLE DB, şi la bazele de date eterogene (din punctul de vedere al modelului de date), folosind driverele ODBC. Se poate conecta şi la fişierele plate, fişierele *Excel,* precum şi la proiectele ***Analysis Service***. Un pachet SSIS poate, de asemenea, copia tabelele dintr-o bază de date în alta.

Un pachet SSIS poate fi creat în două moduri:

- Aplicând ***SQL Server Data Tools***, care în funcţie de configuraţia de instalare poate conţine modulul ***Integration Services Project*** - un instrument grafic destinat utilizatorilor mai experimentaţi şi folosit pentru consolidarea, transformarea şi integrarea datelor eterogene recepţionate din diverse surse.
- Aplicând asistentul SQL Server de import-export. Prin intermediul asistentului, datele pot fi importate, exportate şi transformate din următoarele surse: ***ODBC Data Sources***, ***SQL Server Databases***, tabelele electronice ***Excel***, fişierele ***Access,*** fişierele plain text şi altele.

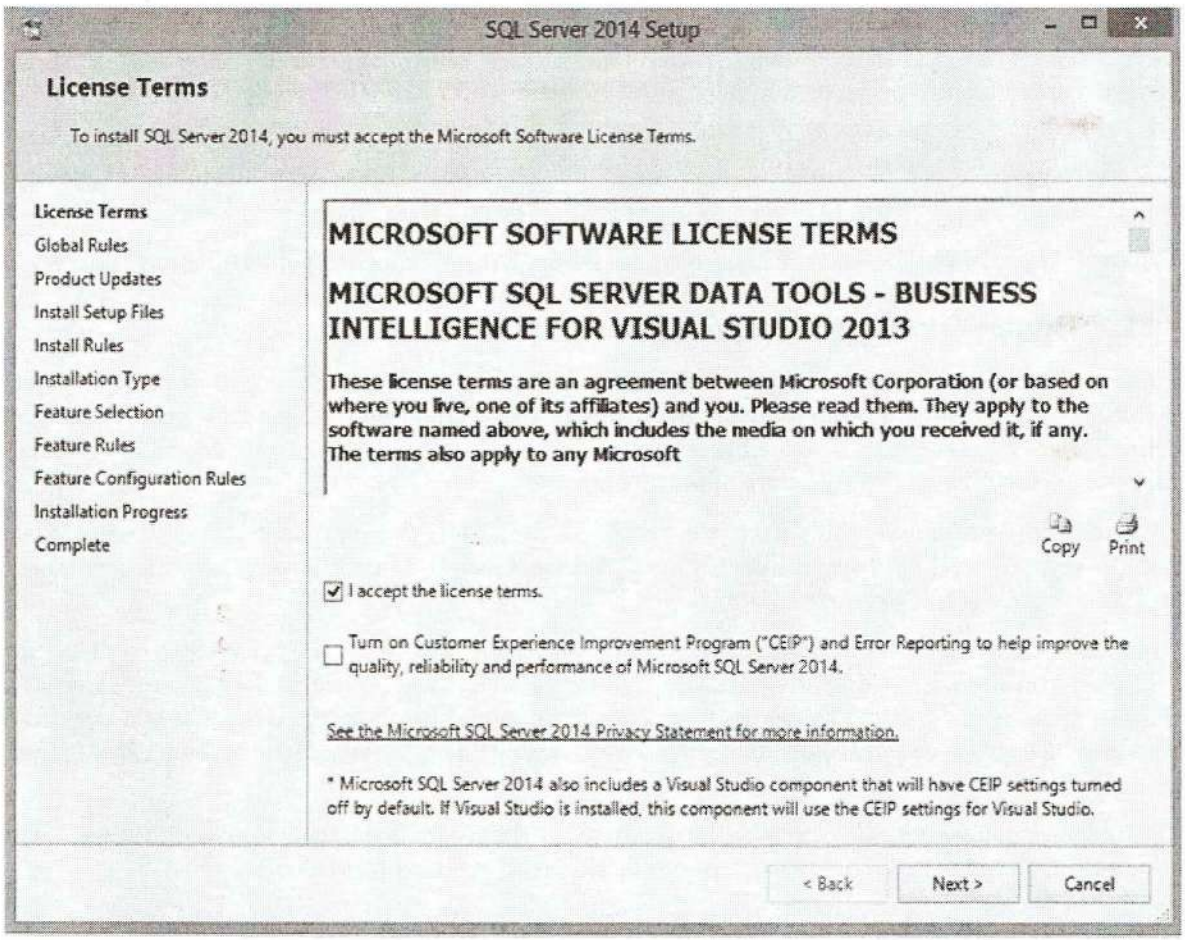
11.2. Instalare SQL Server Data Tools (SSDT)

SQL Server Data Tools (SSDT) reprezintă o extensie a mediului Visual Studio, care conţine şabloanele şi componentele necesare pentru dezvoltarea soluţiilor BI. Spre deosebire de versiunile

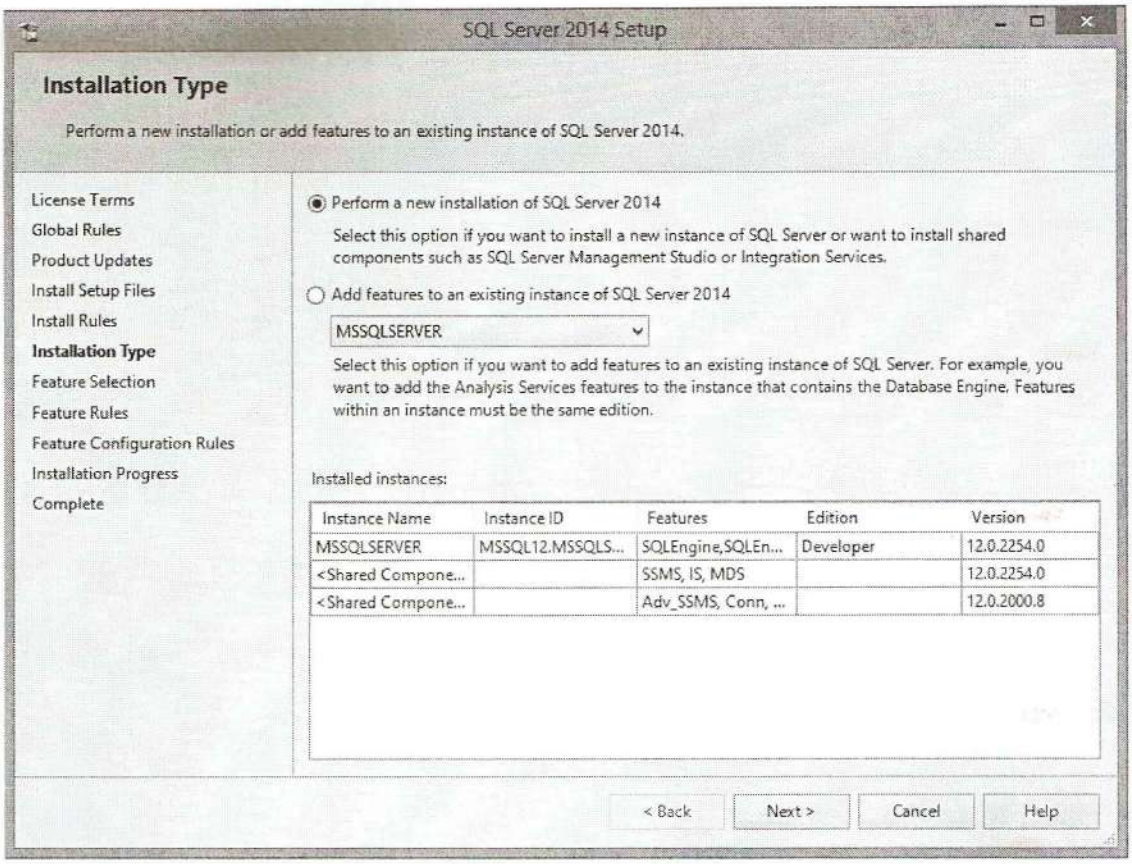
precedente, în versiunea SQL Server 2014, Data Tools nu se instalează împreună cu serviciile SQL Server şi, prin urmare, instrumentul Data Tools trebuie instalat separat.

Pentru a instala SSDT, se parcurg paşii de mai jos. Kit-ul de instalare pentru SQL Server 2014 poate fi descărcat pe adresa: http://www.microsoft.com/en-us/download/details.aspx?id=42313

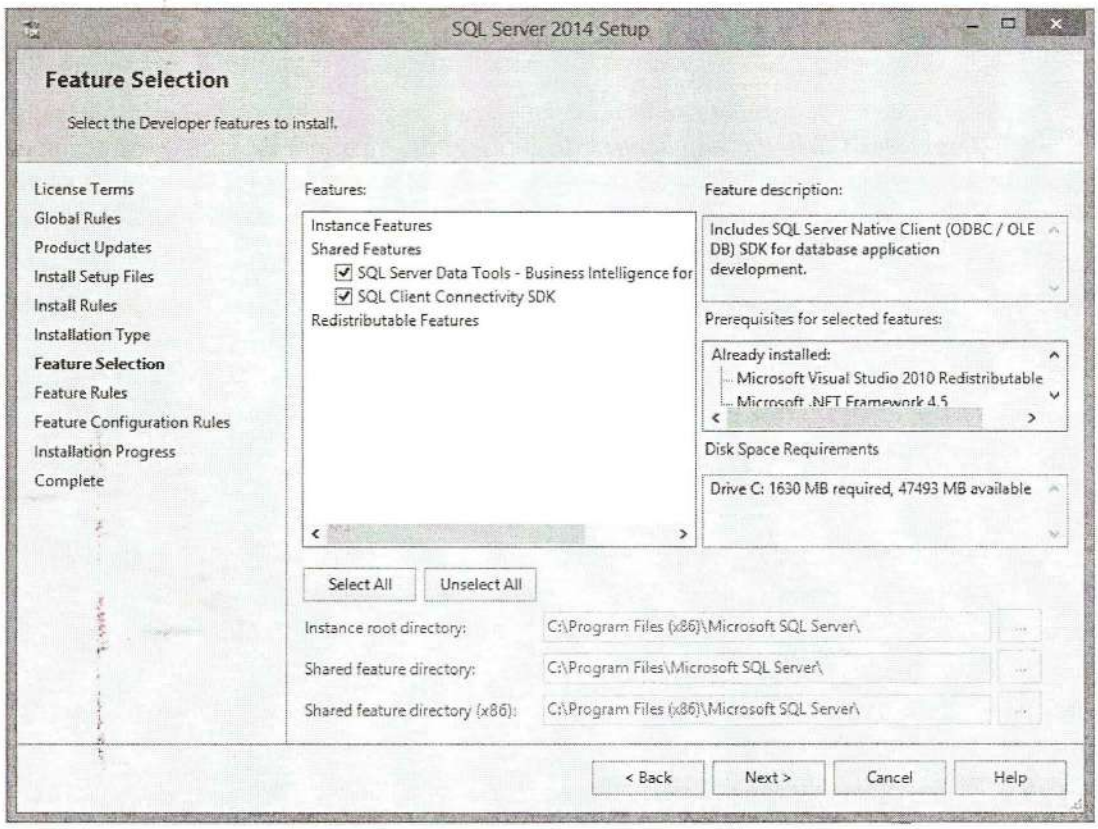
1. Pentru a porni instalarea se execută un dublu-click pe fişierul descărcat. Ca urmare, se lansează asistentul de instalare.
2. Primul pas al asistentului va fi pagina ***License Terms.*** Pentru a instala Data Tools, este necesar să se accepte condiţiile de licenţă prezentate în fereastra din dreapta. Se activează radio-butonul ***I accept the licence terms,*** apoi se apasă **Next**, pentru a continua instalarea sau **Cancel** pentru a întrerupe instalarea.



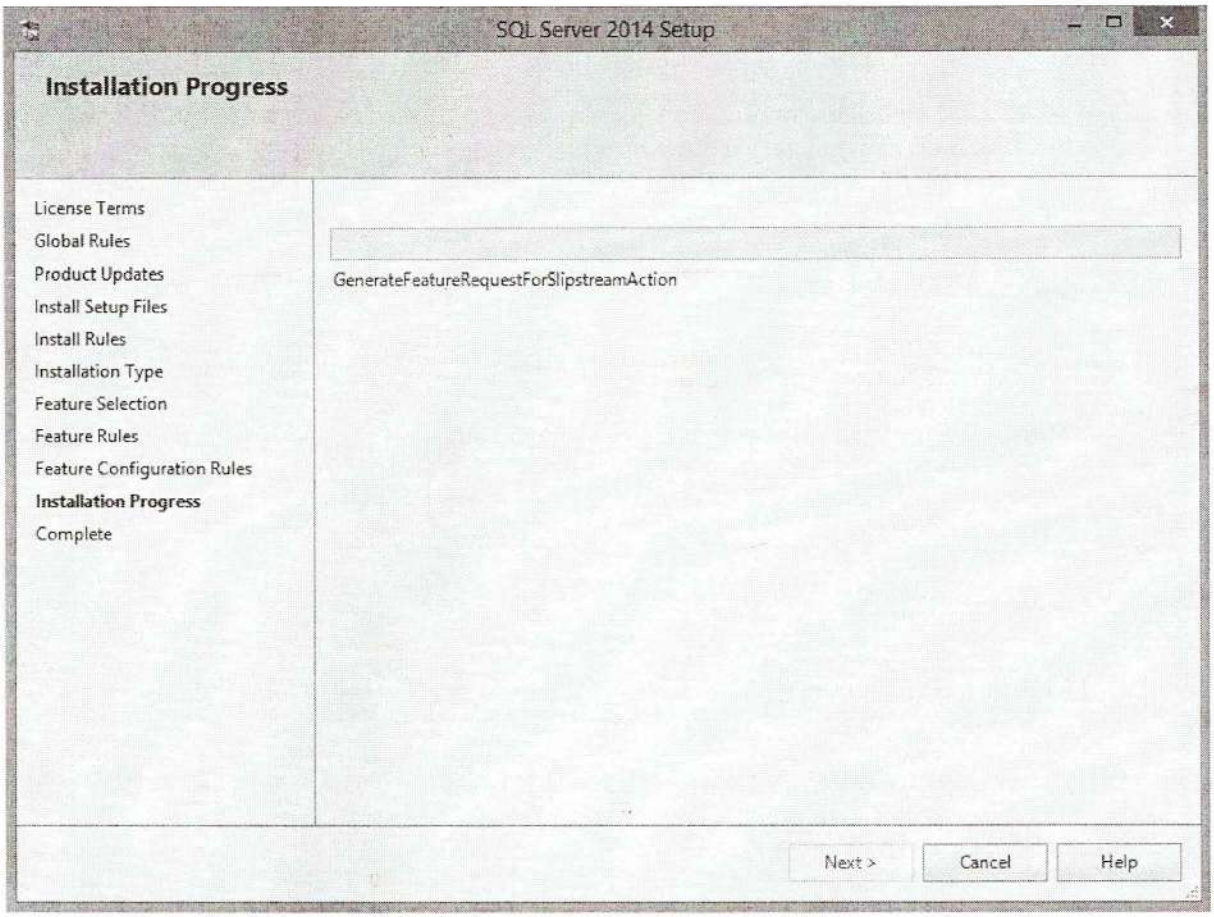
3. Paşii ***Global Rules, Product Updates, Install Setup Files, Install Rules*** sunt similari celor descrişi în capitolul 1 şi sunt parcurşi de asistent, în mod automat, cu condiţia că toate regulile de instalare sunt validate. Dacă vreo regulă are statutul ***Failed,*** atunci instalarea se opreşte până la corectarea cauzei ce a provocat eşecul.
4. În fereastra ***Installation Type,*** se alege opţiunea ***Perform a new installtion of SQL Server 2014*** şi se apasă butonul *Next*. Nu va instala o nouă instanţă SQL Server, ci doar componentele noi Visual Studio, care vor fi selectate la pagina următoare.



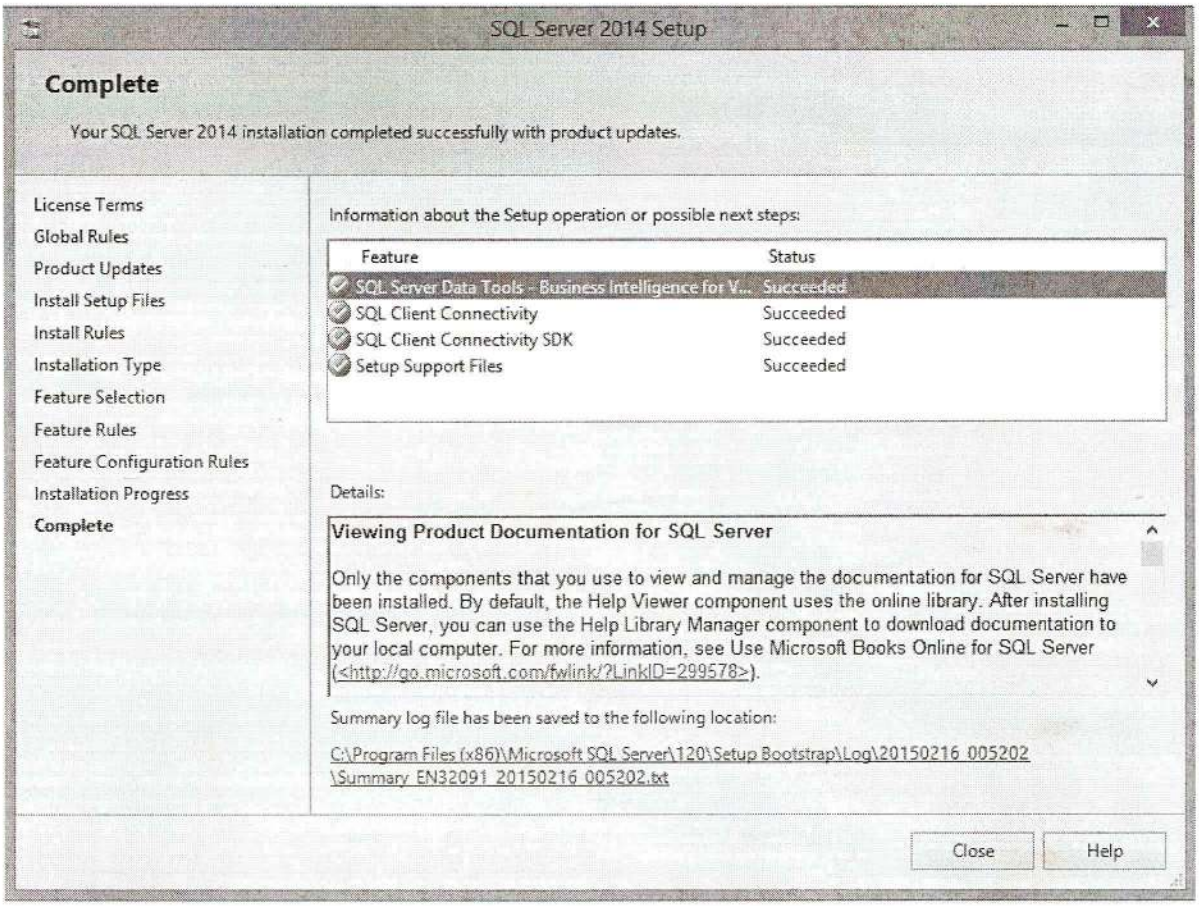
5. În fereastra ***Feature Selection***, se aleg componentele ce vor fi instalate în analogie cu pagina similară din capitolul 1. În cazul nostru, se apasă butonul **Select All** şi apoi, **Next**.



6. Paşii ***Feature Rules, Feature Configuration Rules*** sunt parcurşi, în mod automat, în cazul în care asistentul va valida toate verificările şi, prin urmare, pagina ***Installtion Progress*** se lansează în mod automat.



7. După ce procesul de instalare se termină cu succes, apare pagina ***Complete*** cu mesajul: „***Your SQL Server 2014 installation completed successefully***", iar mai jos se afişează rezumatul stării de instalare pentru fiecare componentă *Data Tools*. În partea de jos a ferestrei, se oferă un link spre fişierul *log* care conţine informaţii sumare despre procesul de instalare. Pentru a închide asistentul de instalare, se apasă butonul **Close.**



Uneori, pentru pornirea corectă a componentelor *Data Tools*, după instalare, se recomandă restartarea calculatorului.

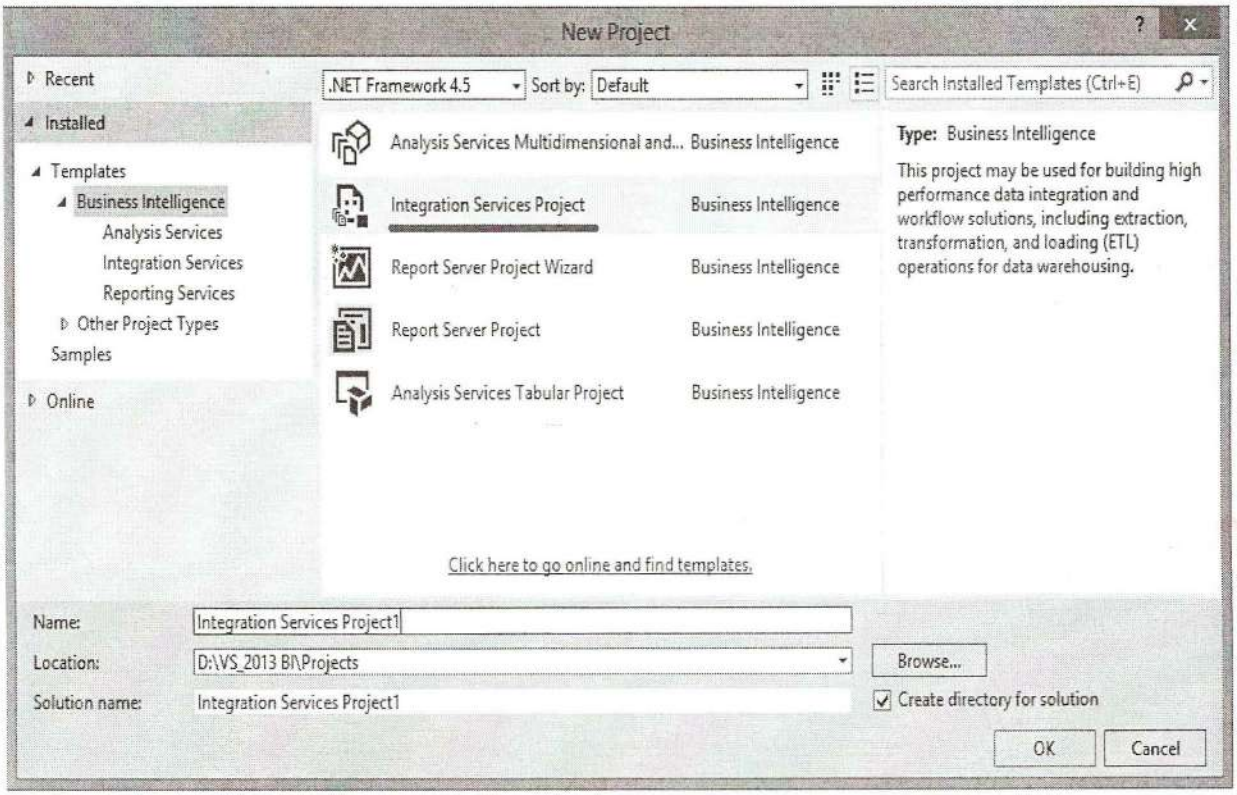
11.3. Gestiunea datelor, aplicând SSIS Package Designer

11.3.1. Crearea şi exportarea pachetelor

Se va crea un pachet SSIS, utilizând ***SSIS Designer***. Pachetul va efectua exportarea unui tabel din baza de date ***calculatoare***, într-o bază de date numită ***tranf***.

Mai întâi, trebuie creată baza de date ***transf***. După crearea acesteia, se parcurg următorii paşi:

1. Se lansează ***Start Screen -> Apps. -> Microsoft SQL Server 2014 -> SQL Server Data Tools for Visual Studio 2013***. În fereastra ***Visual Studio Start Page***, se alege opţiunea ***New Project...***
2. În fereastra apărută, se selectează opţiunea ***Integration service project***, în câmpul de editare ***Name:*** se specifică numele proiectului, iar în câmpul ***Location:*** - locaţia fizică a fişierelor.



3. În acest punct, se va deschide fereastra ***SSIS Designer***, care reprezintă mediul constructorului de pachete al ***Serviciilor de Integrare***.

   În partea de sus a paginii, se găsesc o bară de meniuri şi o bară instrumente. Mai jos, în partea stângă, se află un câmp de obiecte ***SSIS Toolbox,*** care conține obiectele corespunzătoare sarcinilor ce trebuie executate de un pachet de integrare.

   În partea dreaptă, se află fila ***Package.dtsx Design***, care reprezintă câmpul principal de lucru. Fila ***Package.dtsx Design,*** la rândul său, conține cinci file:

   - ***Control Flow***

     Se utilizează pentru construirea, ordonarea şi setarea sarcinilor (inclusiv sarcinii fluxului de date) care asigură funcționalitatea pachetelor, containerelor ce definesc structura pachetelor şi constrângerilor de precedență ce conectează containerele şi sarcinile într-un flux de control. Meniul de context disponibil pe suprafața ***Control Flow*** permite adăugarea notațiilor textuale, setarea punctelor de depanare, mărirea şi micşorarea schemei pachetului. Meniul de context disponibil pe sarcinile individuale permite executarea separată a sarcinilor, fără să fie rulat întregul pachet.

   - ***Data Flow***

     Se utilizează pentru îmbinarea într-un flux de date a surselor care extrag date, a transformărilor care modifică şi agreghează datele, a destinațiilor care încarcă datele, precum şi a căilor care se conectează la intrările şi ieşirile componentelor fluxului de date. Meniul de context al mediului de proiectare ***Data Flow*** permite, de asemenea, adăugarea adnotărilor textuale. Meniul de context disponibil pe căile de conexiunile componentelor fluxului de date admite configurarea ***Data Viewers*** pentru vizualizarea datelor care trec prin fluxul de date.

   - ***Parameters***

     Parametrii Serviciilor de integrare (SSIS) permit atribuirea valorii proprietăților în pachete în momentul executării pachetelor. Fila ***Parameters*** se utilizează pentru crearea şi gestiunea parametrilor la nivelul proiectelor sau la nivelul pachetelor.

- ***Event Handlers***

  Se utilizează pentru setarea fluxurilor de operații care vor fi executate ca un răspuns la evenimentul pachetului. De exemplu, poate fi elaborat un program de tratare a evenimentelor care ar expedia un mesaj pe poștă, în cazul eșecului unei sarcini.

- ***Package Explorer***

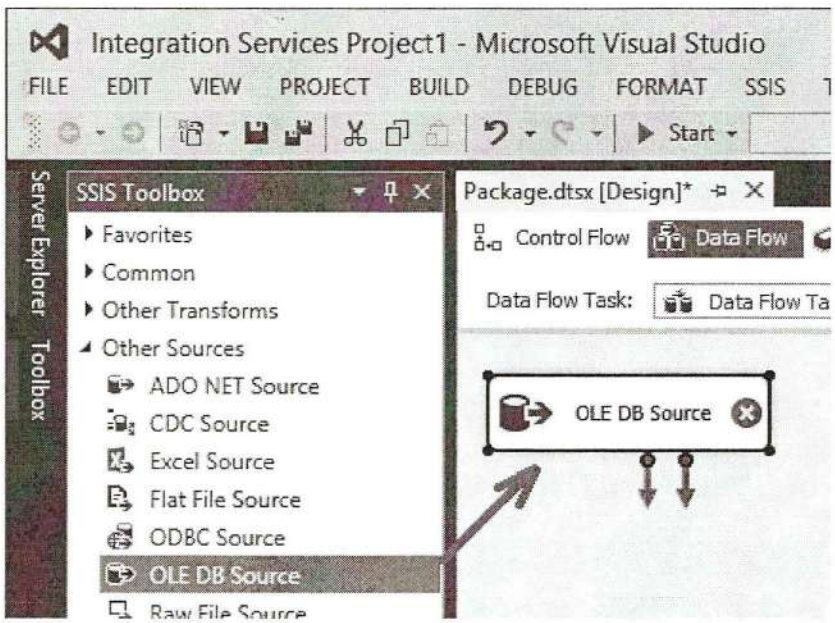
  Se utilizează pentru a obține o vizualizare ierarhică a tuturor elementelor dintr-un pachet, cum ar fi: configurațiile, conexiunile, procesoarele de evenimente, obiectele executabile precum sunt sarcinile și containerele, furnizorii de jurnale, constrângerile de precedență și variabilele. Dacă un pachet conține o sarcină de fluxuri de date, tab-ul ***Package Explorer*** include un nod care conține o viziune ierarhică a componentelor fluxului de date.

În partea dreaptă de sus a constructorului, se află fereastra ***Solution Explorer*** care oferă o viziune organizată a proiectelor și fișierelor acestora, precum și accesul la comenzile ce pot fi aplicate, O bară de instrumente asociată cu această fereastră prezintă comenzile frecvent folosite pentru obiectele selectate din arbore.

În partea dreaptă, mai jos de fereastra ***Solution Explorer,*** se află pagina ***Properties*** care permite vizualizarea sau modificarea proprietăților asociate proiectelor create, fie în fereastra ***Properties,*** fie în caseta de dialog ***Project Property Pages***.

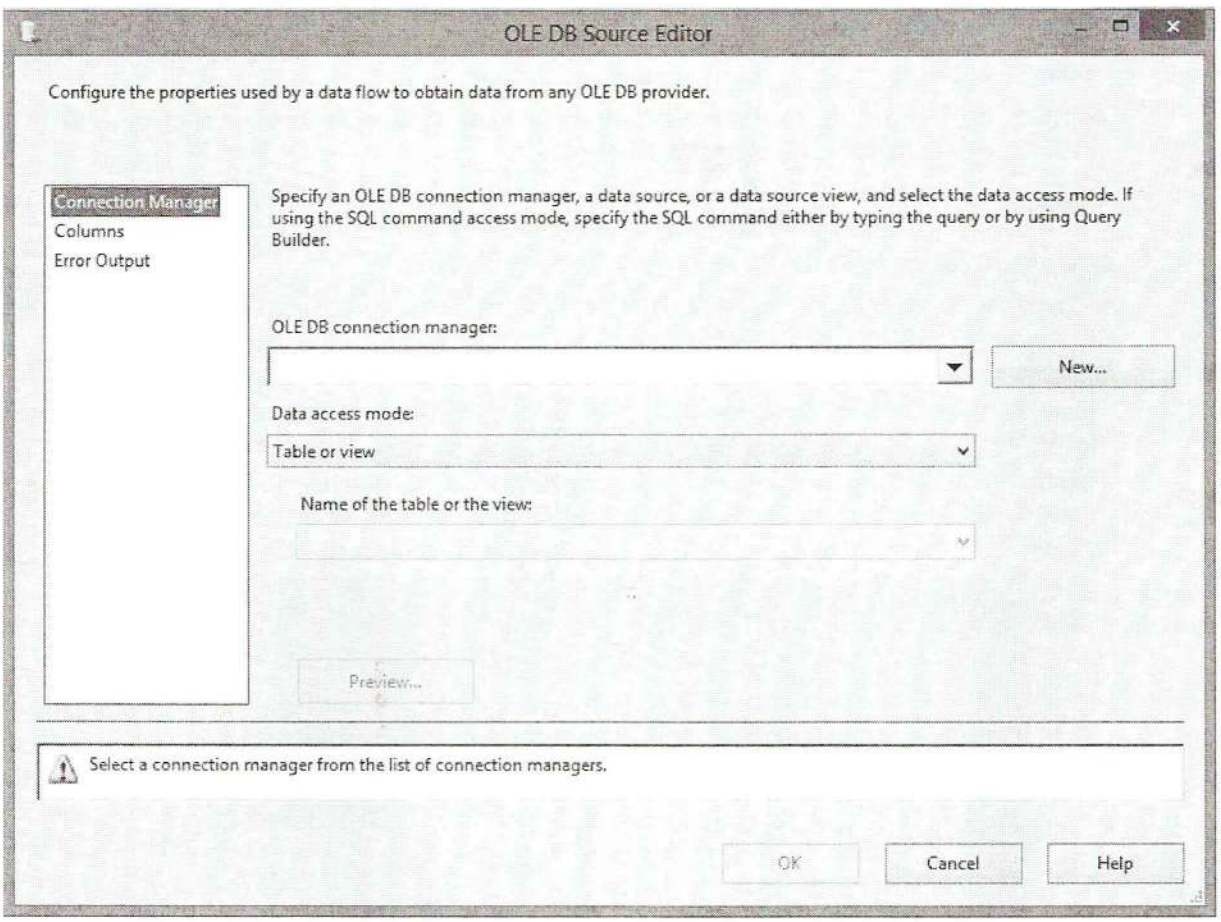
4. Deoarece este nevoie să se exporte date, adică să se creeze niște fluxuri de date, se activează fila ***Control Flow,*** în ***SSIS Toolbox*** se face un click pe ***Data Flow Task***, apoi acest obiect este glisat în aria de lucru.

   Acum, în fluxul de date creat, trebuie sa fie definite intrările și ieșirile. Pentru a intra în corpul fluxului de date, la următorul pas, se face un click pe fila ***Data Flow***. Apoi se marchează obiectul ***OLE DB Sourse***, care se găsește pe bara de instrumente în grupul ***OtherSources***. Acest obiect este glisat în aria de lucru, precum e arătat în următoarea figură:

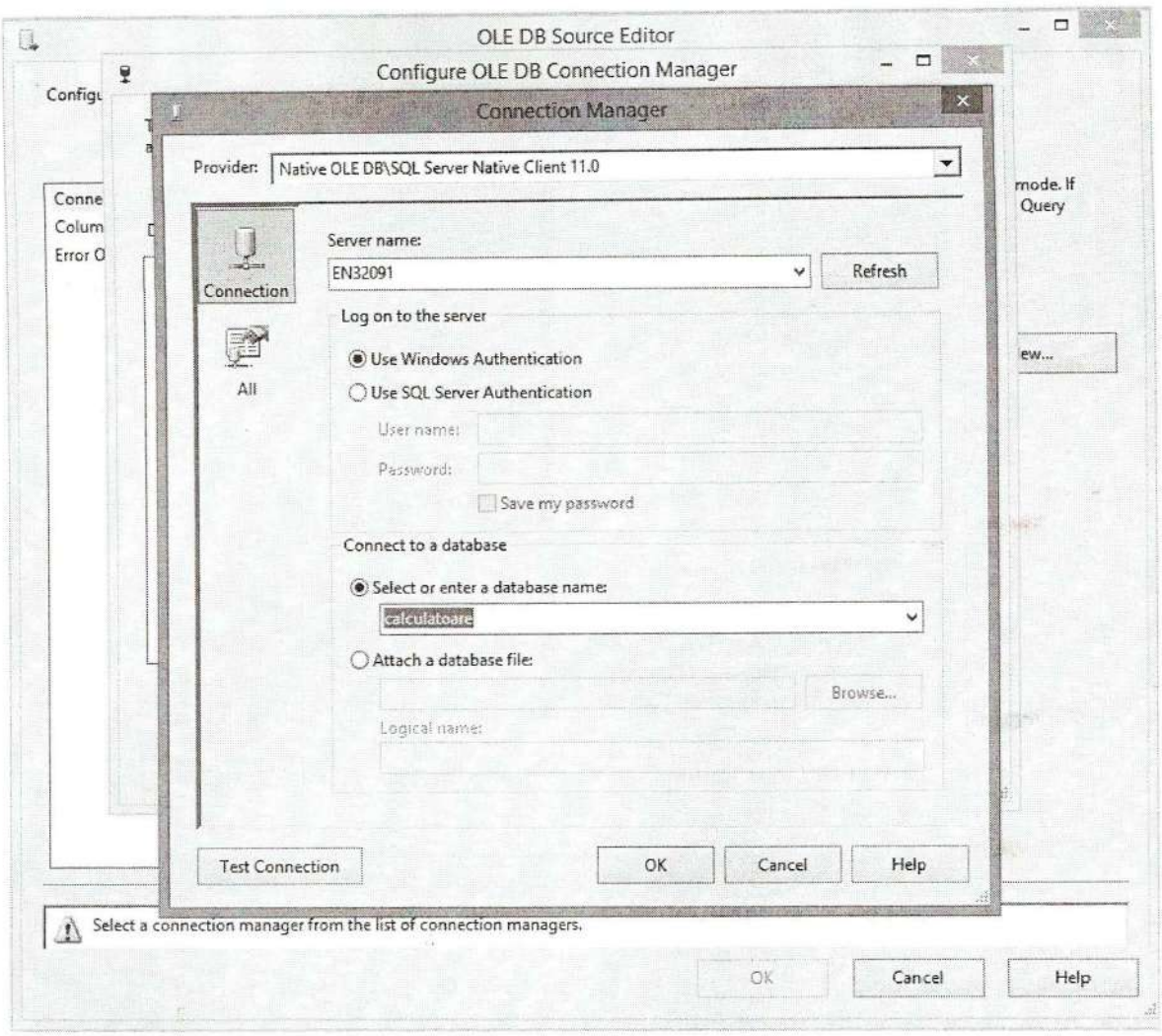


5. Se aplică un click-drept pe obiectul nou în aria de lucru, în meniul de context, se alege ***Edit***. În consecință, pe ecranul calculatorului, va apărea fereastra ***OLE DB Source Editor***.

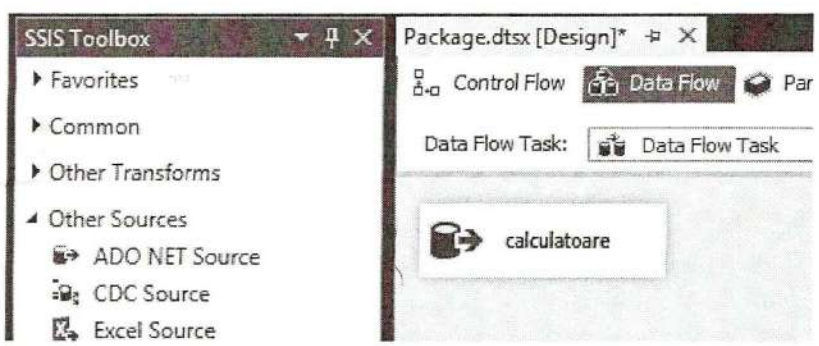
6. Pagina ***Connection Manager*** a ferestrei de dialog ***OLE DB Source Editor*** se utilizează pentru setarea conexiunii OLE DB la sursa de date. Tot în această fereastră este posibilă selectarea tabelelor sau viziunilor din baza de date.
   - În câmpul ***OLE DB connection manager***, dintr-o listă se selectează un gestionar de conexiune existent sau se creează o conexiune nouă, apăsând butonul **New**
   - În câmpul ***Data access mode*** se specifică metoda de selectare a datelor din sursă.
   - Cu ajutorul butonului **Preview** pot fi vizualizate rezultatele. Pot fi vizualizate până la 200 de înregistrări.



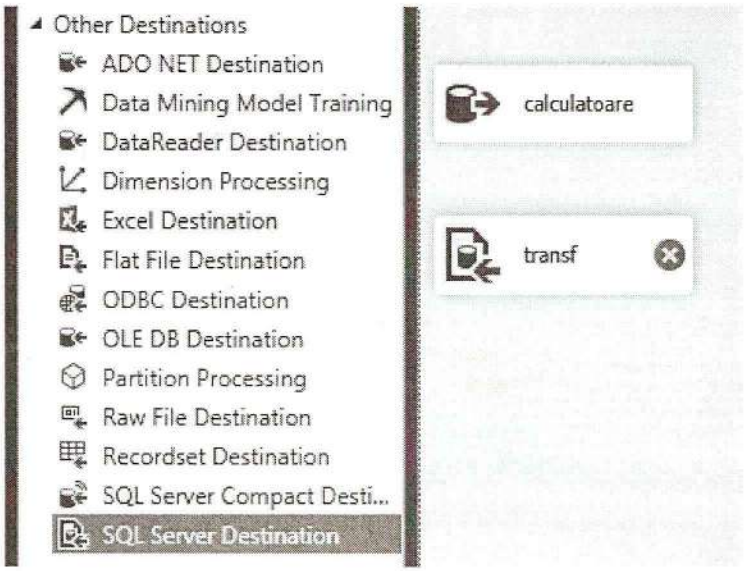
Se acționează butonul **New** din dreapta câmpului ***OLE DB connection manager.*** În fereastra deschisă ***Configure OLE DB Connection Manager,*** pentru a crea o nouă conexiune, se apasă butonul **New....** Apare cutia de dialog ***Connection manager***.



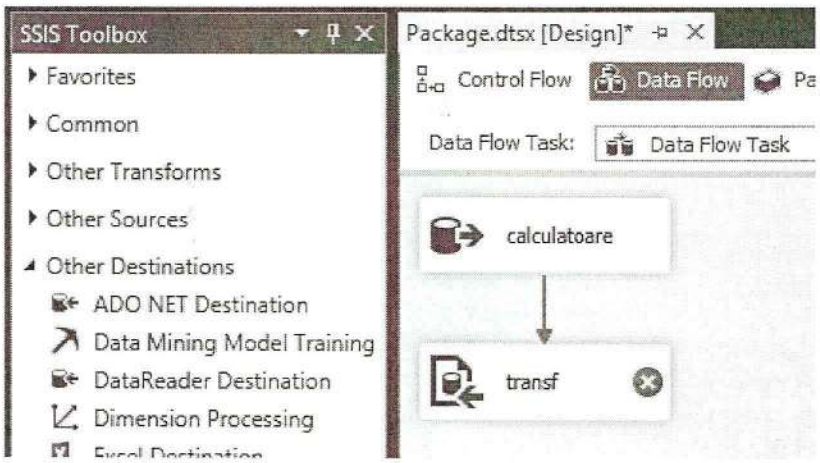
7. În câmpul de editare ***Server name***: se inserează numele serverului la care se doreşte conectarea. În domeniul ***Log on to server***, se alege sistemul de parolă sau de autentificare Windows, sau forma de autentificare a sistemului SQL Server. În cazul dacă se alege ultima opțiune, se inserează numele utilizatorului şi parola acestuia. În domeniul ***Connect to a database***, se alege numele bazei de date sau se ataşează fişierul bazei de date. După setările necesare, se apasă butonul **Test Connection**. În cazul când conexiunea este testată cu succes, se apasă **OK**.
8. În fereastra ***OLE DB Sourse Editor***, în câmpul ***Name of the table or the view***, se selectează tabelul ***pc_uri***, apoi se apasă butonul **OK**.
9. Se redenumeşte obiectul din aria de lucru (***Data flow)*** în ***calculatoare***.



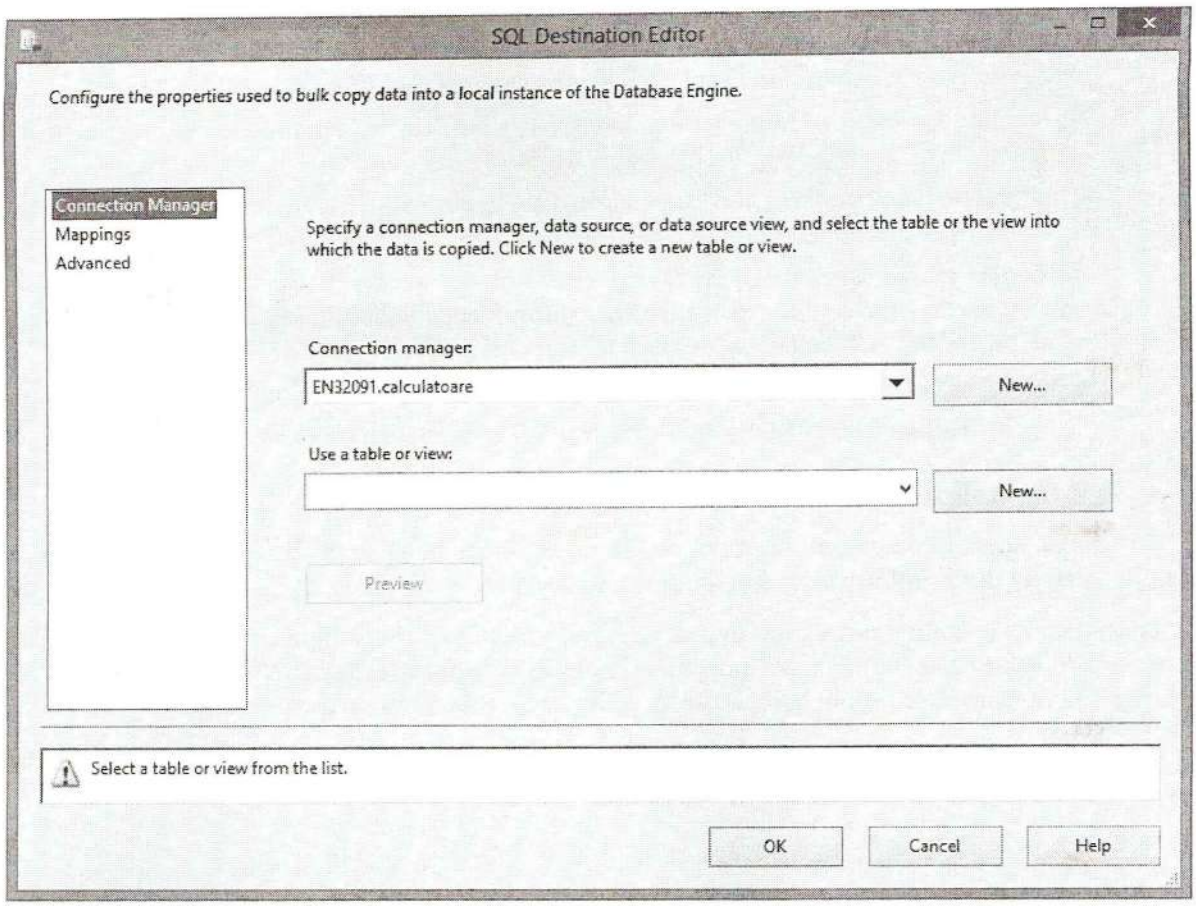
10. În continuare, se creează o conexiune nouă. Pentru aceasta, obiectul ***SQL Server Destination*** care se află în grupul de obiecte ***Other Destinations,*** se glisează spre aria de lucru. Se specifică numele preconizat pentru obiectul nou, adică ***transf***.



11. Se selectează obiectul ***calculatoare*** şi se glisează săgeata albastră spre ***transf***, după cum se vede în fereastra de mai jos.

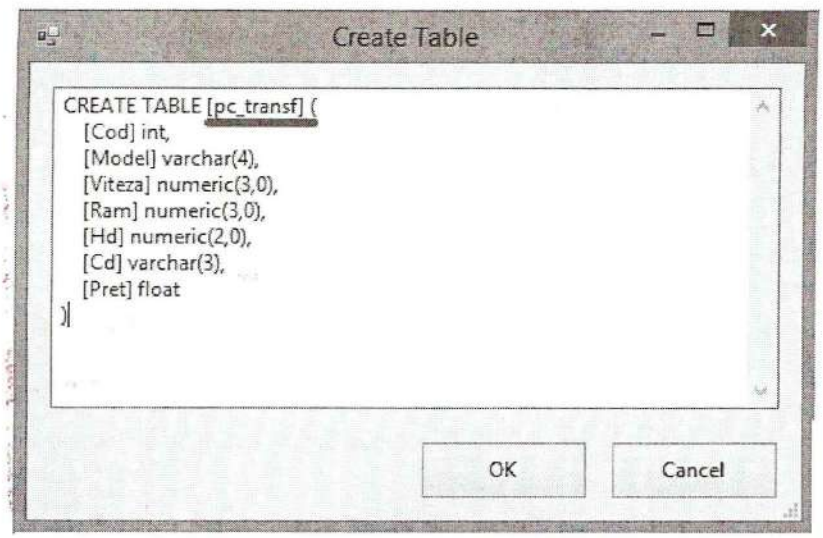


12. Apoi se face click-drept pe obiectul ***transf***, în meniul de context se alege ***Edit***. Astfel, se va deschide fereastra ***SQL Destination Editor***.



13. În fereastra apărută, în primul rând, în căsuța de editare ***Connection manager,*** se setează obiectul-destinație. Procedura este asemănătoare cu cea pentru setarea proprietăților bazei de date-sursă ***(OLE DB Source Editor)***. Astfel, în câmpul de editare ***OLE DB connection manager,*** se alege baza de date destinație. În cazul nostru, aceasta va fi baza de date ***transf***. Apoi, se apasă butonul **New...**din dreapta câmpului de editare ***Use a table or the view***.

    Mai întâi de toate, în cutia de editare apărută (***Create Table***), trebuie specificat numele tabelului-destinație. Se va tipări numele – ***pc_transf***. Sistemul aduce automat toate câmpurile tabelului-sursă. Unele dintre câmpuri, dacă e necesar, pot fi excluse. E destul, pentru aceasta, să fie eliminate din structura afișată. Apoi, se face un click pe butonul **OK**.



14. Este important ca câmpurile-sursă să fie mapate corect în câmpurile-destinație – pagina ***Mapping***. Dacă toate setările sunt corecte, se apasă butonul **OK**.
15. În calitate de ultim pas în crearea unui pachet SSIS, se apasă butonul de pe bara de instrumente de generare a pachetului nou, **Start**.

    Dacă generarea pachetului a avut loc cu succes, în partea inferioară a ferestrei ***Visual Studio*** apare un mesaj informativ despre crearea reușită a pachetului nou.

    Se poate verifica dacă tabelul ***pc_transf*** a fost creat cu succes. Pentru aceasta, în ***SQL Server Management Studio***, în fereastra ***Object explorer*** se extinde baza de date ***transf*** și se face un click pe ***Tables***. Aici trebuie să se vadă tabelul nou creat - ***pc_transf***.

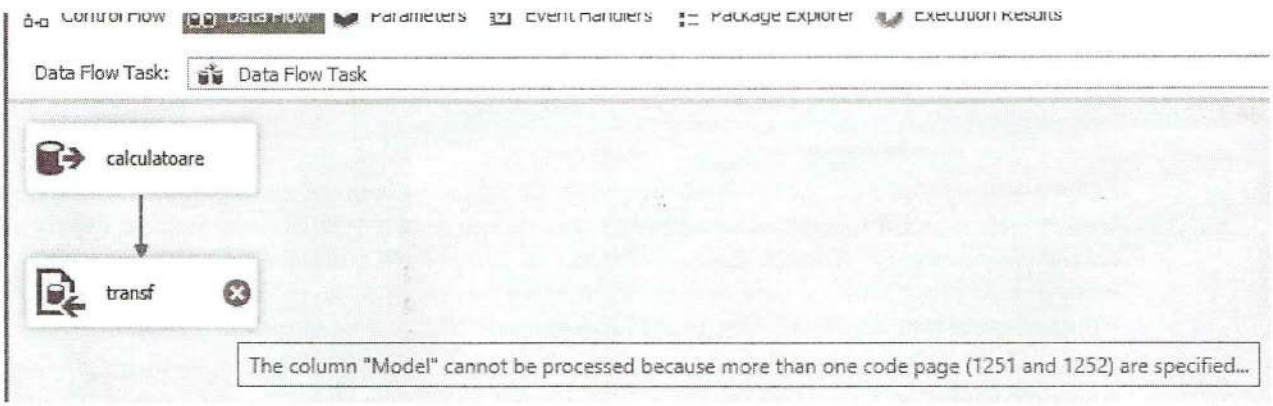
    Dacă se dorește, pentru a vizualiza datele transportate, se poate deschide tabelul ***pc_transf***.

    Trebuie menționat că într-un singur pachet pot fi create mai multe conexiuni.

11.3.2. Depanarea pachetelor SSIS

Integrarea datelor, de regulă, necesită o înțelegerea mai bună a proceselor de transformare și convertire a datelor dintr-un format în altul, dintr-o formă in alta.

După crearea și configurarea fiecărui task sistemul validează corectitudinea acestuia, dacă validarea nu este reușită task-ul rămâne marcat cu cerculeț roșu. Aceasta însemnă că pachetul nu este configurat corect. Apropiind mausul de taskul marcat roșu se poate vedea mesajul de eroare respectiv.

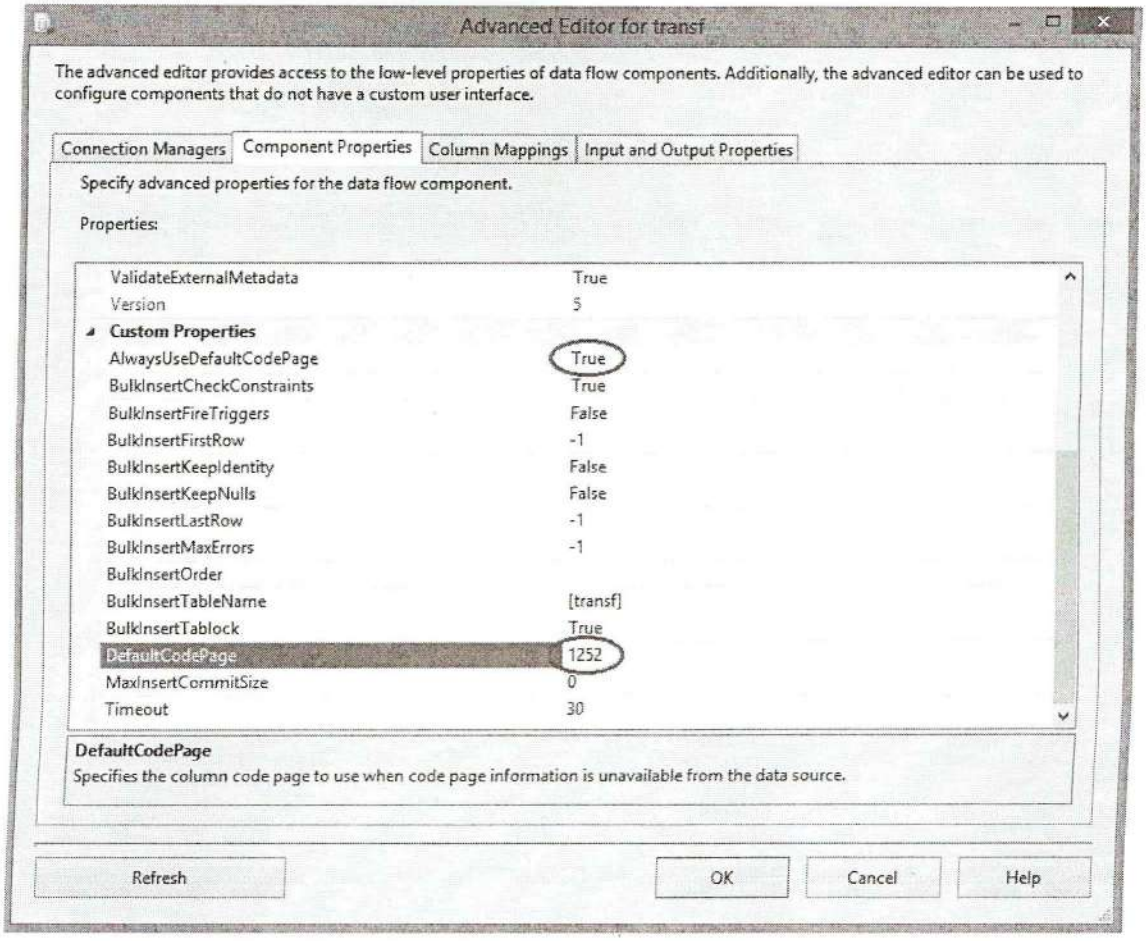


Același mesaj poate fi obținut sub o forma mai utilă daca în bara de meniu ***Visual Studio*** se alege ***View***, apoi in meniul derulant se alege ***Error List***. În consecință, jos apare un tabel cu lista de erori și avertizări generate de sistem în procesul validării pachetului.

De exemplu, mesajul de eroare din imagine informează că setul de caractere în sursa de date nu este compatibil cu setul de caractere la destinație.

Pentru a soluționa problema concretă din exemplu, se parcurg următorii pași:

1. Se face un click drept pe taskul ***transf***.
2. În meniul de context apărut se alege opțiunea ***Show Advanced Editor***.
3. În fereastra Advanced Editor se alege fila Component Properties:



4. In tabelul ***Properties*** se editează 2 câmpuri:
   - Câmpului ***AlwaysUseDefaultCodePage*** din grupul ***Custom Properties*** i se atribuie valoare ***True***. Valoare ***True*** indică faptul că dacă codificarea paginilor din coloana de tip caracter nu este specificată ei i se atribuie valoarea implicită.
   - Cumpului ***DefaultCodePage*** i se atribuie codul *1252*.
5. Se apasă butonul **OK**.

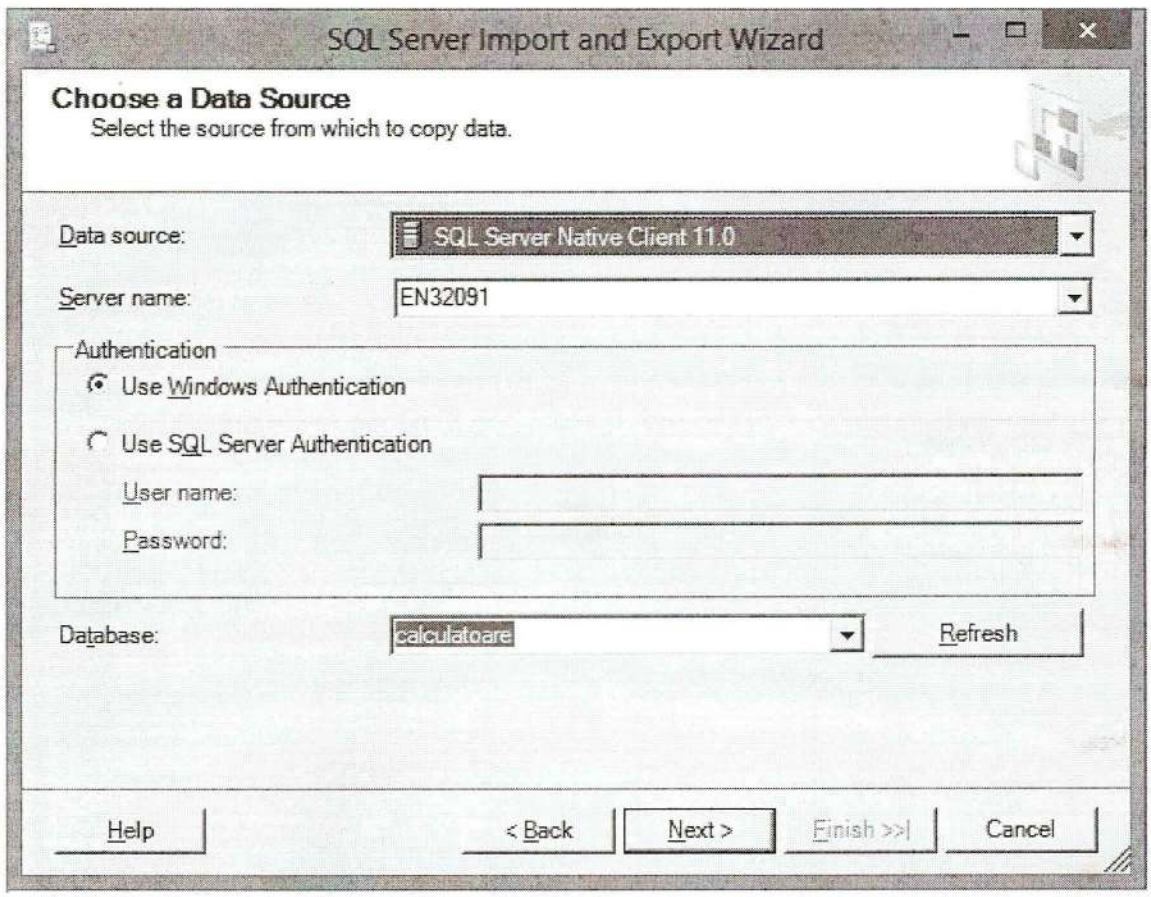
11.4. Exportarea datelor cu ajutorul asistentului

În continuare, se va examina cum se exportă datele sistemului SQL Server (în cazul altui tip de fişiere), utilizând asistentul de exportare. Procesul este foarte simplu:

1. La început, se extinde serverul respectiv, se extinde dosarul ***Databases,*** se face un click cu butonul drept al mouse-ului pe baza de date - sursă de date destinată pentru export.

   În cazul nostru, iarăşi se alege baza de date ***calculatoare.***
2. În meniul de context, care a apărut, se aplică un click pe opţiunea, ***Tasks,*** apoi pe ***Export Data..***

   În consecinţă, se deschide prima fereastră ***Welcome to SQL Server Import and Export Wizard*** a asistentului de copiere şi transformare a datelor între sursele de date şi destinaţiile susţinute. Se apasă butonul **Next**.
3. La următorul pas apare fereastra ***Choose a Data Source***, unde se specifică sursa de date care trebuie copiată.



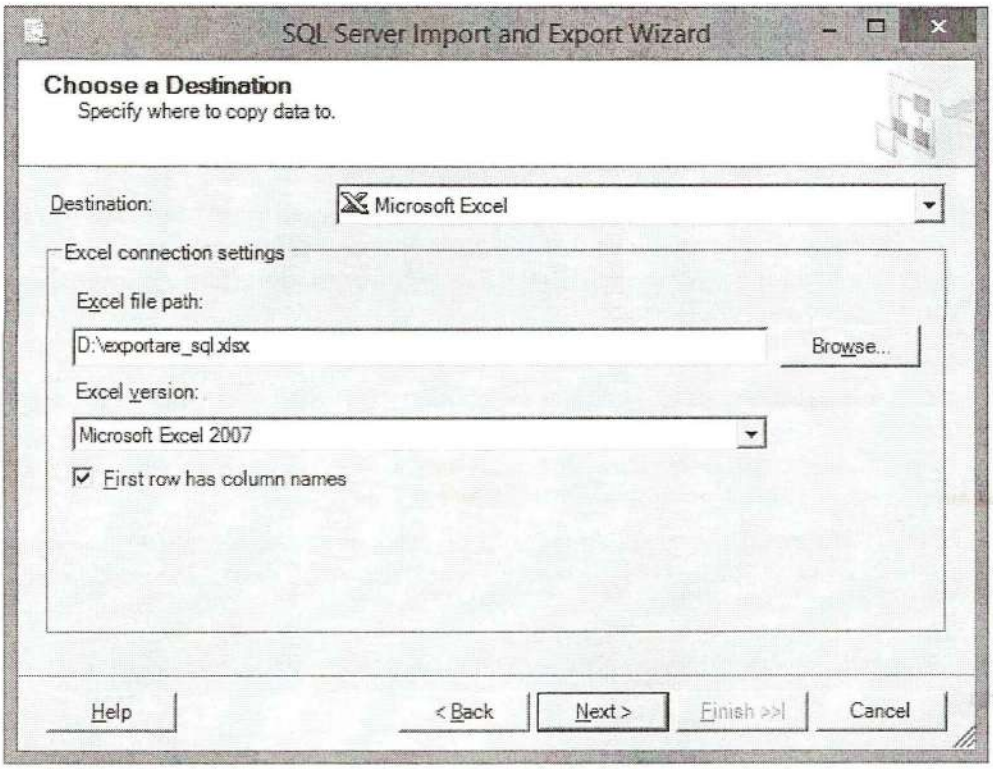
4. În meniul derulant al câmpului ***Data Source***, se alege furnizorul de date care se potriveşte formatului de stocare a datelor la sursă. Poate fi mai mult decât un singur furnizor disponibil pentru datele-sursă.

   În cazul nostru, rămâne opțiunea implicită – ***SQL Server Native Client 11.0.*** În câmpul ***Server name***, se selectează serverul care conține baza de date (tabelul), din care se vor exporta datele. Apoi, se selectează tipul de autentificare, activând unul din radiobutoane. Dacă se alege tipul de autentificare ***SQL Server Authentication***, trebuie să se completeze câmpurile ***User Name*** şi ***Password.***

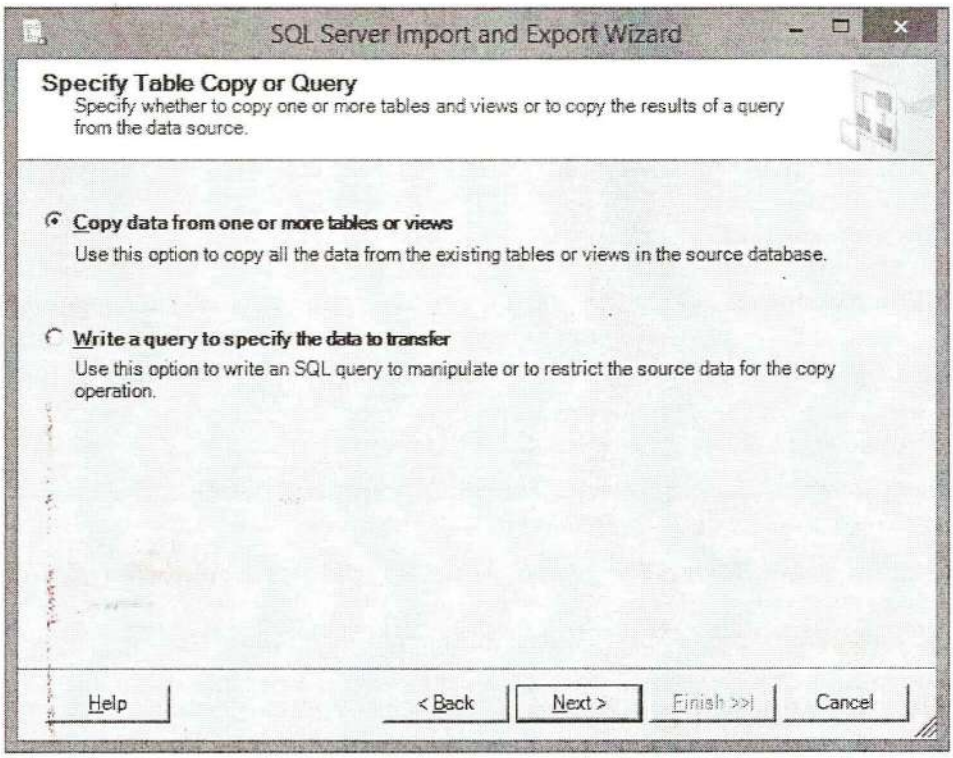
   În câmpul ***Databases*** se alege baza de date care conține tabelul ce trebuie exportat. În cazul nostru, va fi selectată baza de date *calculatoare.* Se face un click pe butonul **Next.**

5. În fereastră apărută ***Choose a Destination***, se selectează destinația exportului. În meniul derulant al câmpului ***Destination***, se selectează furnizorul de date care se potriveşte formatului de stocare a datelor la destinație.

   În cazul nostru, se alege opțiunea ***Microsoft Excel***. Să se observe că fereastra este modificată în mod automat, ca în figura ce urmează:



6. În această fereastră modificată, în câmpul de editare ***Excel file path***, se specifică calea și numele fișierului care va păstra datele exportate. În cazul nostru, se selectează directoriul-rădăcină ***D:*** cu numele fișierului ***exportare_sql***. În câmpul de editare ***Excel version*** se alege versiunea fișierului-destinație (***Excel***) care va fi utilizată. Apoi, se face un click pe butonul **Next**. Se va deschide următoarea fereastră:

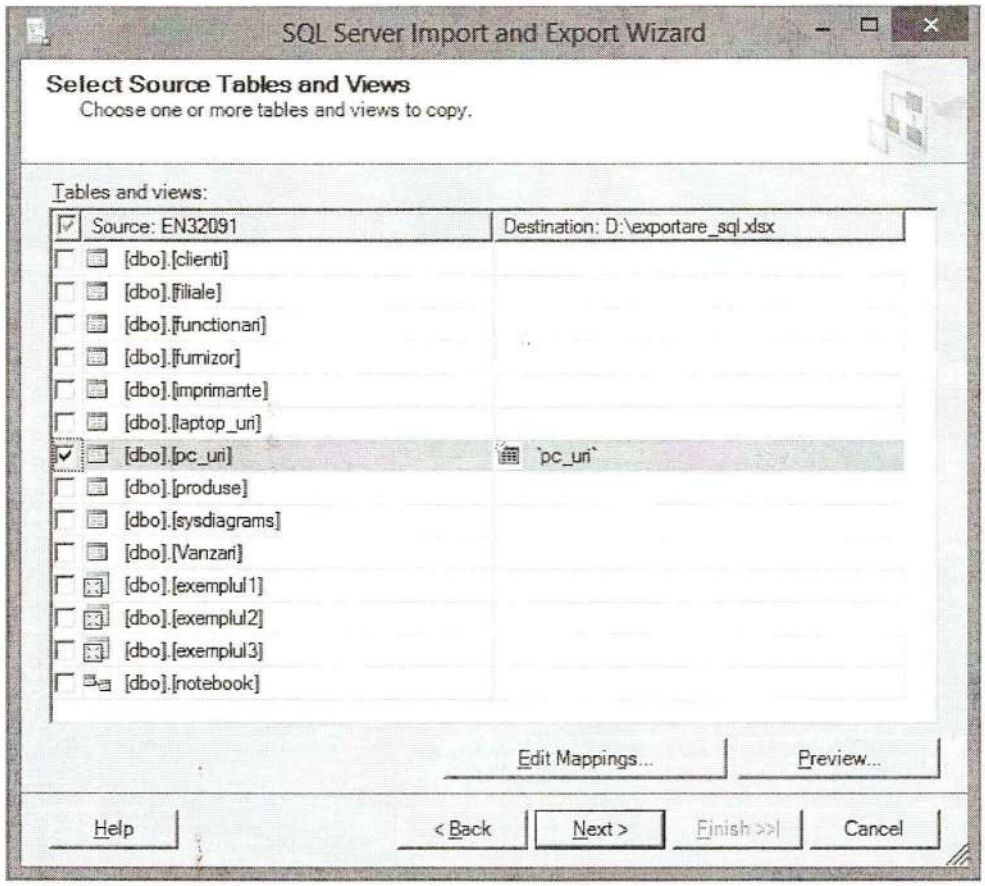


7. În fereastra ***Specify Table Copy or Query***, pentru a specifica modul de copiere a datelor, se selectează una din cele două radiobutoane. Pentru a selecta obiectele existente din baza de date care trebuie copiate, poate fi utilizată o interfață grafică sau poate fi folosită o instrucțiune Transact-SQL cu scopul de a crea o interogare mai complexă.
   - Opțiunea ***Copy data from one or more tables or views***

     Copie câmpurile selectate din tabelele şi viziunile-sursă în destinația sau destinațiile specificate, utilizând caseta de dialog ***Select Source Tables and Views***. Această opțiune se foloseşte pentru copierea de la sursă a tuturor datelor, fără filtrare sau ordonare a înregistrărilor.
   - Opțiunea ***Write a query to specify the data to transfer***

     Construieşte instrucțiuni SQL pentru căutarea înregistrărilor, utilizând caseta de dialog ***Provide a Source Query***. Această opțiune se foloseşte pentru modificarea sau restricționarea sursei de date în timpul operațiilor de copiere. Sunt disponibile pentru copiere numai liniile care se potrivesc criteriilor de selecție.

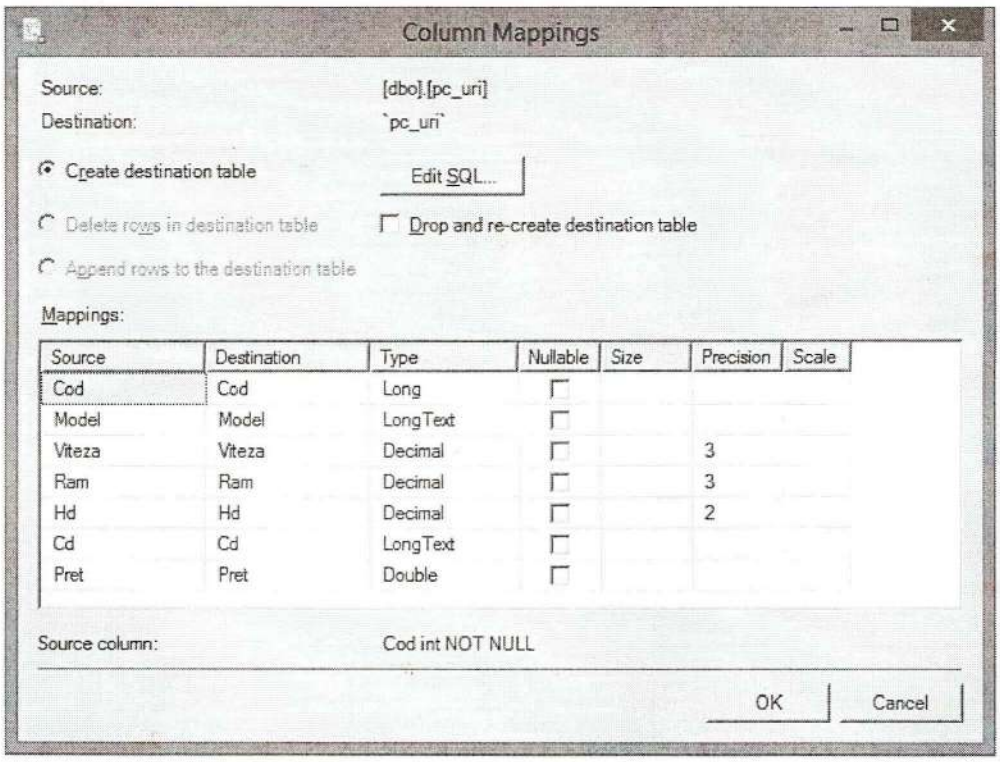
   În cazul nostru, rămâne marcată opțiunea implicită, pentru transportarea tuturor înregistrărilor tabelului. Apoi se aplică un click pe butonul **Next**, după care se deschide următoarea fereastră:



8. În fereastra ***Select Source Tables and Views,*** se specifică tabelele şi viziunile care vor fi copiate de la sursă la destinație.
   - În coloana ***Surce,*** utilizând căsuțele din stânga, se selectează tabelele şi viziunile destinate copierii. Dacă se selectează un tabel sau o viziune şi nu se face nici o altă acțiune, schema şi datele din sursă vor fi copiate fără modificări.
   - În coloana ***Destination,*** pentru fiecare tabel-sursă se selectează din lista derulantă tabele-destinație disponibile. În cazul nostru, se marchează tabelul ***pc_uri.***

- Butonul **Preview** activează caseta de dialog ***Preview Data*** care oferă posibilitatea de vizualizare a datelor înainte de a efectua operațiuni de import sau export. În casetă pot fi afișate până la 200 de rânduri de date.
- Prin apăsarea butonului **Edit mappings** se deschide caseta de dialog ***Column Mappings,*** unde pot fi editate proprietățile coloanelor din tabel, care vor fi exportate.

Se apasă butonul **Edit mappings.** În consecință, apare fereastra ***Column Mappings:***



9. Apăsând butonul **Edit SQL...** se deschide o cutie de editare, ***Create Table SQL Statement,*** unde poate fi editată schema tabelului exportat, reprezentată în limbajul SQL.

   În cazul nostru, se lasă totul fără schimbări. Apoi se apasă **OK** și se revine la fereastra-mamă. În continuare, se aplică un click pe butonul **Next**. Ca urmare, asistentul va trece la următorul pas.

10. La această etapă apare fereastra ***Review Data Type Mapping*** care are funcția de revizuire a informației detaliate despre convertirea tipurilor de date care trebuie să o realizeze asistentul pentru a face ca sursa de date să fie compatibilă cu destinația.

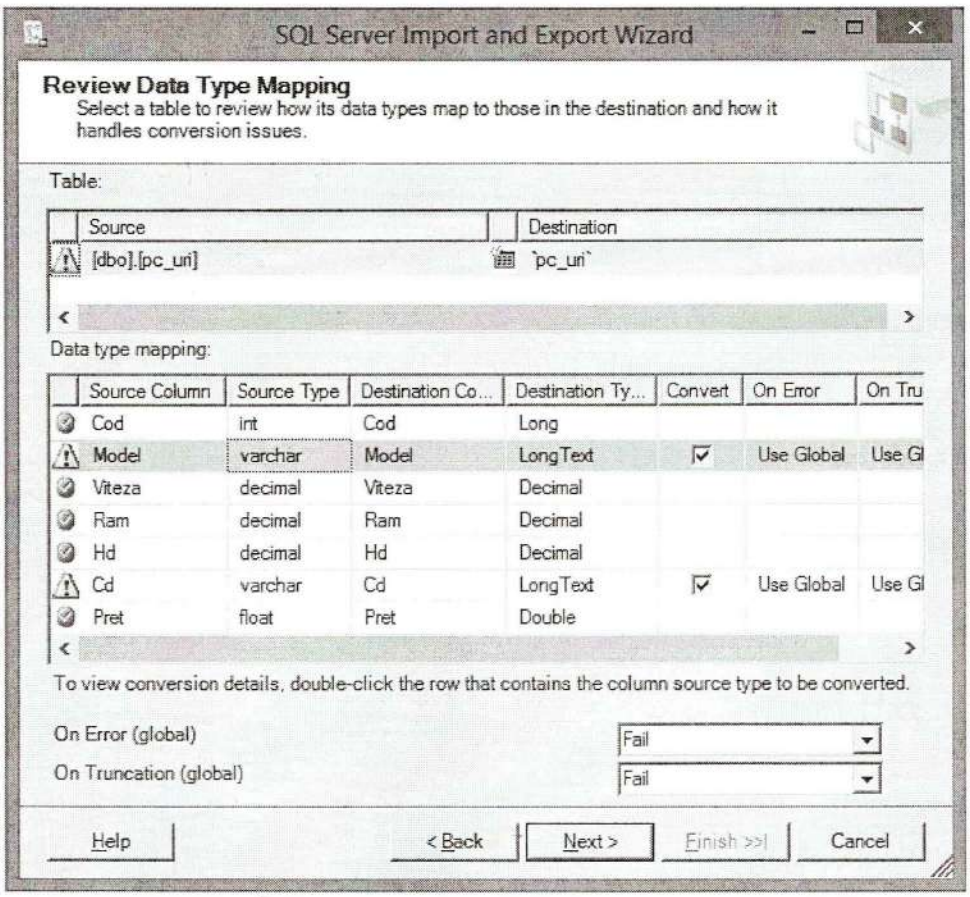
    Această pagină include un set de semne vizuale pentru a distinge conversiile care se succed de conversiile care pot provoca erori sau trunchieri de date. Pentru fiecare conversie se poate decide dacă se acceptă sau nu varianta de conversie sugerată de asistent și se poate specifica modul de tratare a erorilor ce pot apărea.

    În partea de sus a paginii ***Review Data Type Mapping*** se află câmpul ***Table*** care conține lista de tabele care servesc drept sursă de date pentru lista de tabele-destinație respective din partea dreaptă.

    În partea stângă lângă fiecare tabel sursă se află câte un indicator care indică probabilitatea succesului în conversia tipului de date:

    - O pictogramă de culoare verde indică faptul că asistentul așteaptă ca toate datele pentru acest tabel să fie convertite.
    - O pictogramă de culoare galbenă indică faptul că este necesar să se revizuiască conversiile individuale pe care le va efectua asistentul. Pentru revizuirea acestor conversii, se selectează tabelul respectiv și apoi se revizuiesc conversiile pentru coloanele respective în tabelul ***Data type mapping***.

- O pictogramă de culoare roşie indică faptul că asistentul nu este capabil să realizeze unele din conversii şi să păstreze totodată consistenţa datelor exportate.



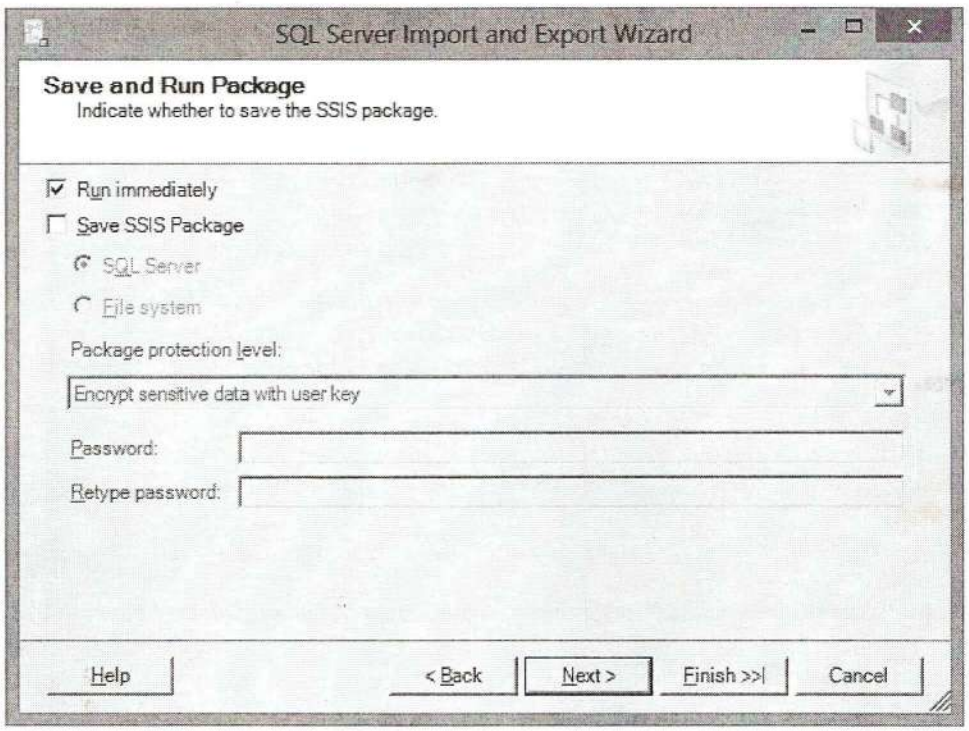
Selectând un tabel sau altul, în gridul ***Data type mapping*** se afişează o listă de coloane destinate exportării. În acest tabel sunt câteva atribute care necesită o atenţie specială:

- În coloana ***Convert*** se specifică dacă conversia planificată ar trebui să continue. Pentru ca asistentul să continue conversia, căsuţa trebuie bifată. Dacă căsuţa se debifează, conversia tipului de date se anulează.
- În coloana ***On Error*** se specifică modul în care asistentul va trata erorile apărute. Sunt posibile trei variante de tratare a erorilor:
  - utilizarea setării ***Use global***;
  - eşuarea în caz de eroare şi stoparea procesului de import/export;
  - ignorarea erorii.
- În coloana ***On Truncation*** se specifică modul în care asistentul va executa trunchierea. Sunt posibile trei variante de tratare a erorilor:
  - utilizarea setării ***Use global***;
  - eşuarea în caz de eroare şi stoparea procesului de import /export;
  - ignorarea erorii.

Pentru vizualizarea detaliilor conversiei ale fiecărui tip de date în parte, se face dublu click pe înregistrarea respectivă a tabelului ***Data type mapping.***

În partea inferioară a ferestrei ***Review Data Type Mapping*** sunt amplasate instrumentele de gestiune a erorilor. Sunt două cazuri de erori: ***On Error (global)*** şi ***On Truncation (global)***. Pentru ambele cazuri, sunt posibile următoarele scenarii de comportament al sistemului: eşuarea şi oprirea procesului de export sau ignorarea erorii şi continuarea procesului de export. Aceste setări se aplică pentru toate conversiile care în tabelul ***Data type mapping*** au selectată valoarea ***Use Global.***

În cazul nostru, se lasă toate setările neschimbate, fiind cele optimale pentru scenariul dat de export. Se apasă butonul **Next.**



11. La această etapă apare fereastra ***Save and Run Package***. Aici se poate decide dacă exportul trebuie să fie executat imediat sau să fie salvat ca un pachet SSIS şi planificat pentru o dată ulterioară. Opţiunea ***Save SSIS Package*** poate avea două acţiuni:
    - ***SQL Server*** - salvează pachetul într-o bază de date ***msdb*** a sistemului ***Microsoft SQL Server.***
    - ***File system*** - salvează pachetul într-un fişier cu extensia ***.dtsx.***

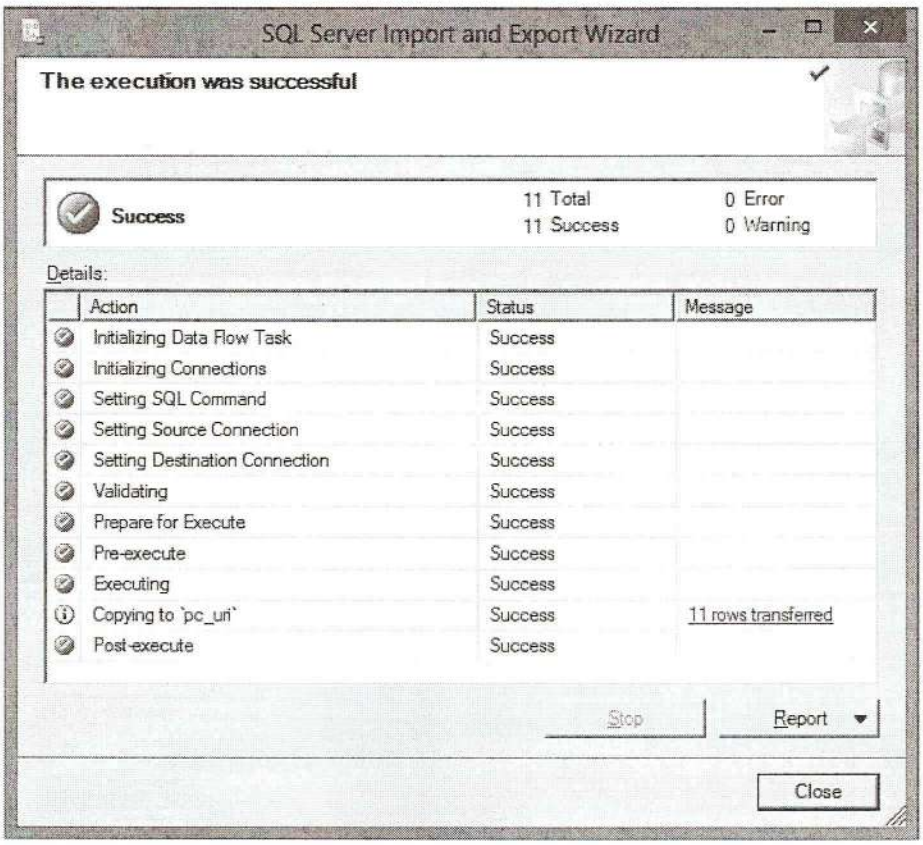
    În cazul în care se selectează opţiunea ***Save SSIS Package,*** devine activ câmpul ***Package protection level*** cu ajutorul căruia se defineşte nivelul de protecţie, fiind accesat în lista derulantă.

    Nivelul de protecţie determină metoda de protecţie, parola utilizatorului sau cheia acestuia, precum şi domeniul de aplicare a protecţiei pachetului. Protecţia datelor poate să includă toate datele sau numai datele sensibile.

    În cazul nostru, rămâne marcată opţiunea de executare imediată şi urmează un click pe butonul **Next**.
12. Următoarea pagină se numeşte ***Complete the Wizard*** şi prezintă un rezumat al operaţiilor realizate. Se acţionează butonul **Finish.**
13. În final, se afişează pagina ***Performing Operation***, unde este vizualizat procesul de export/import şi rezultatele acestui proces. Dacă este necesar, procesul de import/export poate fi întrerupt, apăsând butonul **Stop**. În gridul ***Details***: sunt afişate acţiunile care fac parte din operaţie, statutul lor şi mesajele de informare sau de eroare.

    Apăsând butonul **Report**, raportul despre executarea operaţiei import/export poate fi salvat într-un fişier sau expediat pe e-mail.

Dacă totul este corect, în partea superioară a paginii apare următorul mesaj: ***The execution was successful.*** Se apasă butonul **Close**.



În directoriul ***D:***, va fi creat un fişier Excel, şi acesta va avea atâtea pagini, câte tabele au fost exportate.

11.5. Gestiunea pachetelor SSIS în mediul SQL Server Management Studio

Pachetele SSIS create pot fi păstrate sub forma de fişiere şi apelate de către SQL Server din sistemul de operare sau pot fi integrate în corpul SQL Server şi apelate din baza de date ***msdb*** (în cazul implementării de tip Proiect, proiectele construite sunt stocate în baza de date ***SSISDB***).

Pentru gestiunea şi executarea pachetelor SSIS în ***SQL Server 2014*** este necesară conectarea serviciului ***Integration Services...*** Conectarea serviciului ***Integration Services*** la interfaţa ***SQL Server Management Studio*** poate fi realizată urmărind scenariul:

1. Se deschide SQL Server Management Studio.
2. Pe bara de instrumente a ferestrei ***Object Explorer*** se apasă butonul **Connect.**
3. În meniul derulant apărut se alege opţiunea ***Integration Services.***

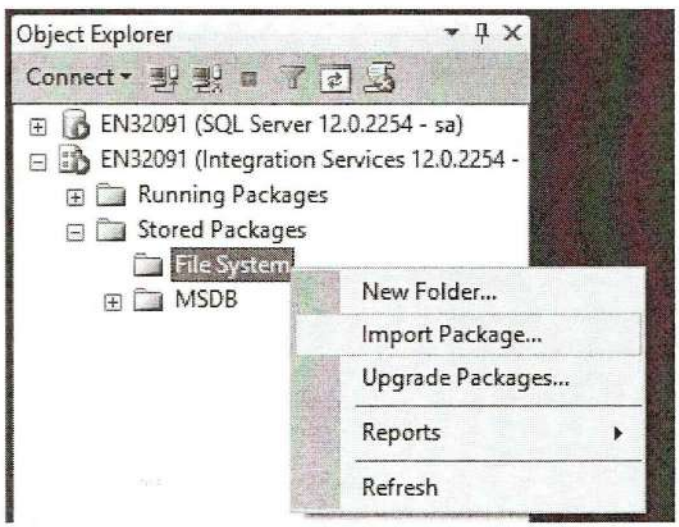
Procedura de autentificare este asemănătoare celei de conectare la ***Database Engine...*** Deja, există un pachet creat şi salvat pe discul rigid cu ajutorul constructorului ***Visual Studio***.

11.5.1. Importarea unui pachet SSIS

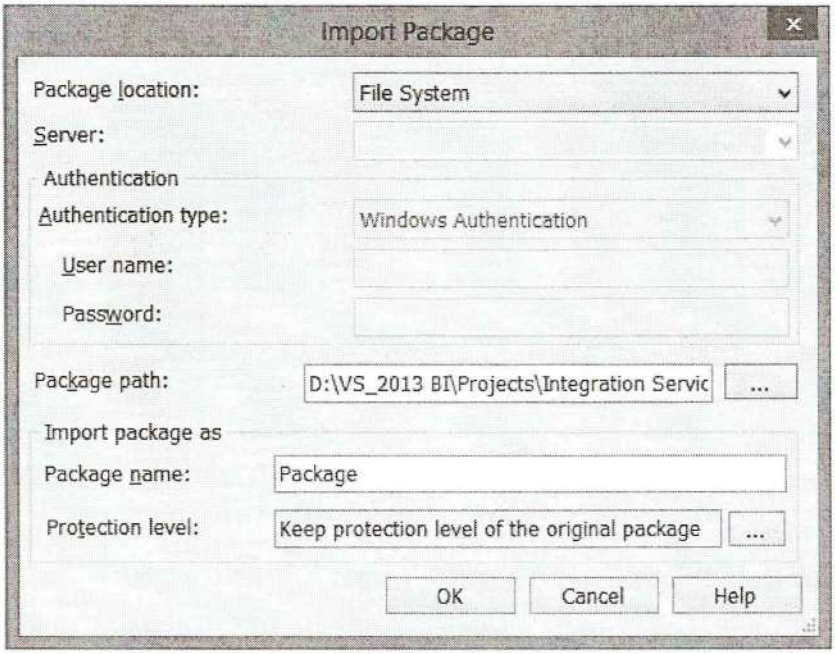
Este o sarcină foarte frecventă pentru administratorii de baze de date pentru a importa pachete SSIS de la o instanță SQL Server sau de la un sistem de fişiere într-o instanța locală de SQL Server. În această secțiune sunt prezentați paşii care trebuie să fie urmați pentru a importa un pachet, utilizând ***SQL Server Management Studio.***

Pentru conectarea şi includerea acestui pachet în sistemul de fişiere sau în ***MSDB***, se parcurg următorii paşi:

1. Se extinde componenta ***Integration Service***.
2. Se extinde dosarul ***Stored Packages***.
3. Se face un click-drept pe dosarul ***File System*** (sau ***MSDB***).
4. În meniul de context apărut, se alege opțiunea ***Import Package...***



Apare caseta de dialog ***Import Package***, care se utilizează pentru importul pachetului construit de ***SQL Server Integration Services*** şi pentru setarea sau modificarea nivelului de protecție al pachetului.

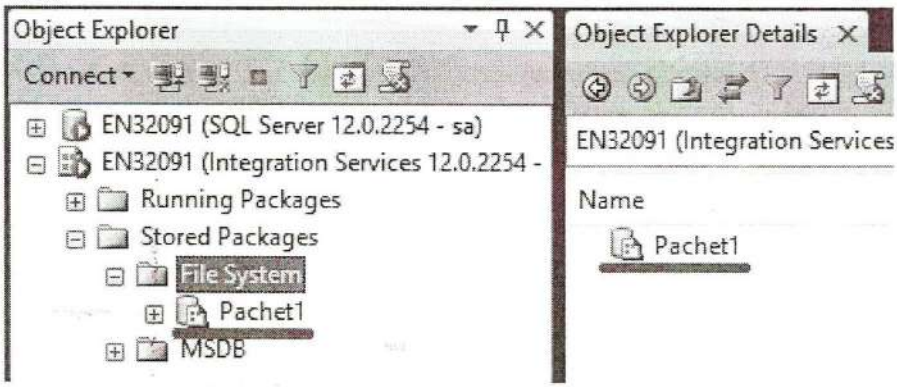


- În câmpul de editare ***Package location,*** se selectează tipul locației de stocare, unde se va importa pachetul. Sunt disponibile câteva variante.

  Se alege File System.
- În caseta ***Package path,*** se specifică calea spre pachet, sau se aplică un click pe butonul **Browse**, și se stabilește locul pachetului.

  În cazul dat, se specifică spre pachetul nostru D:\VS_2013 BI\Projects\Integration Services Project1.
- În câmpul ***Package name,*** în mod opțional, se redenumește pachetul. Numele implicit este numele pachetului importat.

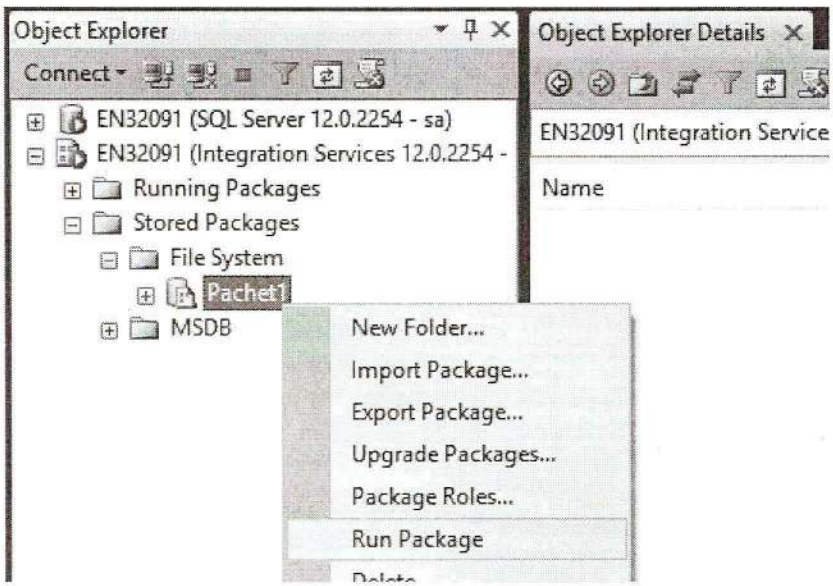
5. Apoi se apasă butonul **OK**. În acest moment, în arborele ***Object Explorer***, a apărut pachetul atașat.



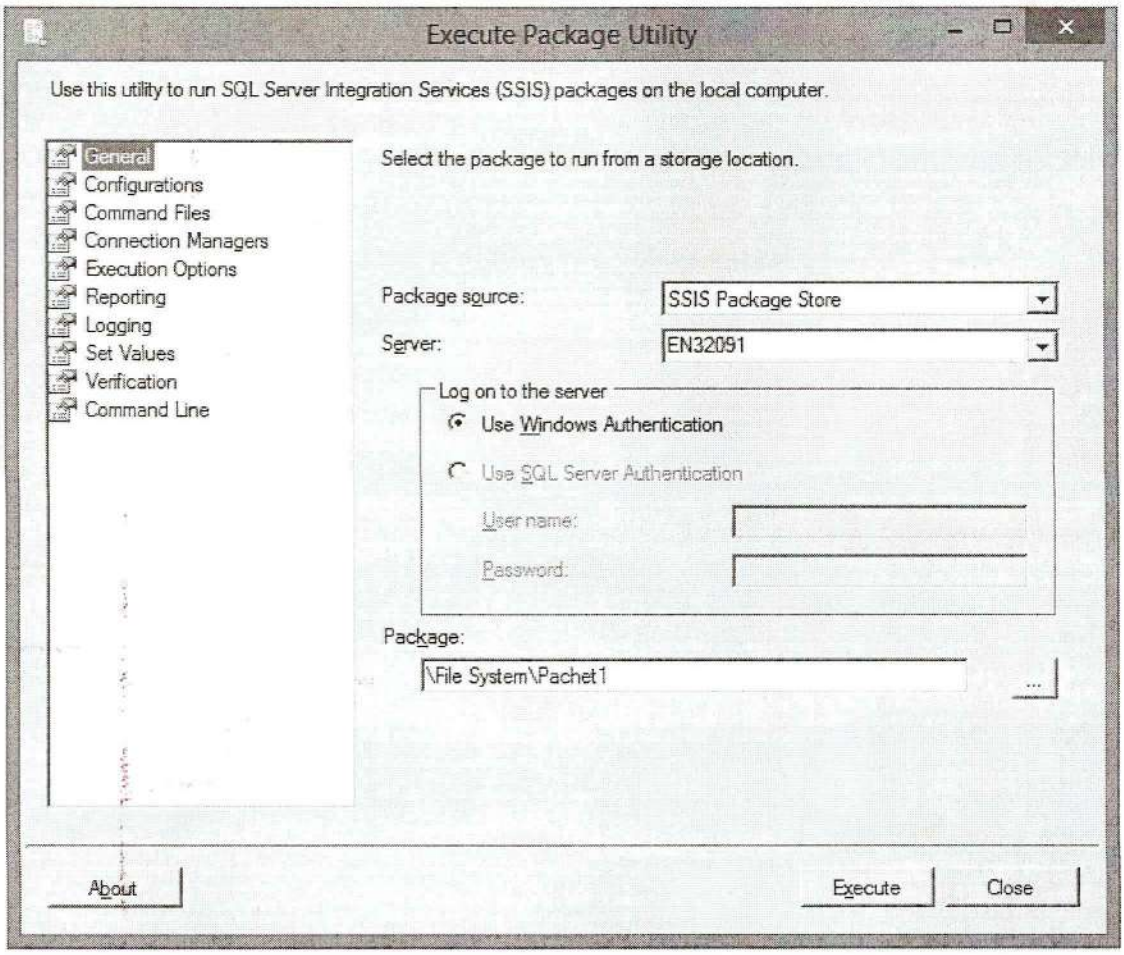
6. Pentru gestiunea pachetului, în ***Object Explorer*** se apasă cu butonul drept al mouse-ului pe pictograma pachetului respectiv.

11.5.2. Executarea unui pachet SSIS

1. Pentru executarea pachetului, în meniul de context se alege opțiunea ***Run Package***:



2. Apare fereastra ***Execute Package Utility*** care este utilizată pentru setarea proprietăților de executare a unui pachet:



3. Se apasă butonul **Execute.** În consecință, apare fereastra ***Package Execution Progress,*** unde sunt vizualizate detaliile executării pachetului.

Întrebări de control:

1. Ce reprezintă *SSIS*? Care sunt componentele şi funcționalitățile acestui serviciu?
2. Particularitățile procesului de exportare a datelor, aplicând ***SSIS Package Designer.***
3. Particularitățile procesului de exportare a datelor cu ajutorul asistentului.
4. Gestiunea pachetelor *SSIS* de SGBD-ul *SQL Server 2014*

Sarcini practice:

1. Să se exporte, utilizând SSIS, rezultatele rezolvării ale două exerciții din capitolul 4 (instrucțiuni *SELECT*), un export fiind realizat cu ajutorul asistentului de import/export, iar altul aplicând ***SSIS Package Designer.*** Datele, în primul caz, trebuie să fie exportate într-o bază de date de tip ***MS Access***, iar în al doilea caz, trebuie să fie exportate în tabelul ***transf.exercitiul1***.
2. Să se importe datele în tabelul ***transf.exercitiul2*** din baza de date Access creată în exercițiul anterior.

12. SERVICII DE CREARE A RAPOARTELOR

Capitolul tratează stadiul actual al serviciilor de creare şi gestiune a rapoartelor în mediul ***SQL Server Data Tools.*** Sunt examinate aspectele referitoare la afişarea datelor în forma unor pagini hipertext, precum şi gestiunea proprietăţilor serviciilor de rapoarte.

Microsoft SQL Server 2014 Reporting Services (SSRS) oferă o gamă completă de instrumente şi servicii pentru crearea, distribuirea şi gestiunea rapoartelor despre datele extrase din baza de date şi pune la dispoziţie o serie de facilităţi ce permit ajustarea şi extinderea funcţionalităţilor de generare a rapoartelor pe cale programabilă [4,5].

SSRS reprezintă o platformă de generare a rapoartelor bazată pe server, care oferă funcţionalităţi comprehensive de generare a rapoartelor pentru o varietate largă de surse de date. Serviciile includ un set complet de instrumente pentru crearea, gestiunea şi expedierea rapoartelor. Interfeţele API permit programatorilor să integreze sau să-şi extindă posibilităţile de procesare a datelor şi rapoartelor în aplicaţiile utilizatorilor. Instrumentele acestor servicii funcţionează în mediile ***Microsoft Visual Studio*** şi ***SQL Server Report Builder*** şi sunt complet integrate cu instrumentele şi componentele sistemului *MS SQL Server*.

Procesarea datelor este distribuită între mai multe componente. Procesoarele central şi cele specializate sunt folosite pentru căutarea datelor, procesarea aspectului raportului, procesarea formatelor de prezentare şi expedierea acestora la destinaţie. Prelucrarea prezentării este decuplată de procesarea datelor şi are loc după ce datele sunt găsite. Acest fapt permite mai multor utilizatori să vizualizeze acelaşi raport simultan în formate proiectate pentru diverse dispozitive sau să modifice print-un singur click formatul de vizualizare a raportului, din HTML în PDF sau Microsoft Excel, sau XML.

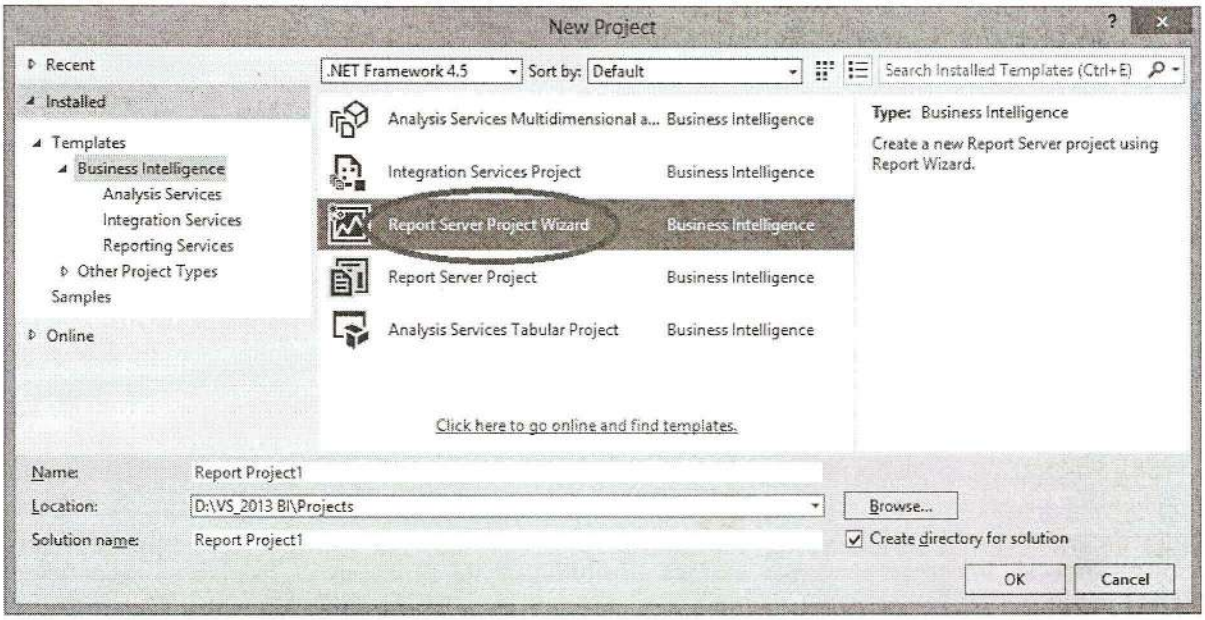
Paginile rapoartelor pot fi create de un asistent sau în regim de constructor şi pot fi plasate pe un server WEB sau vizualizate în reţeaua Intranet. În plus, acestea pot fi vizualizate în formatul ***asp*** cu ajutorul unui navigator. Datele din rapoarte pot fi exportate din formatul ***asp*** în fişiere cu alte extensii, de exemplu ***.txt***, ***.pdf.***

12.1. Gestiunea rapoartelor cu ajutorul asistentului

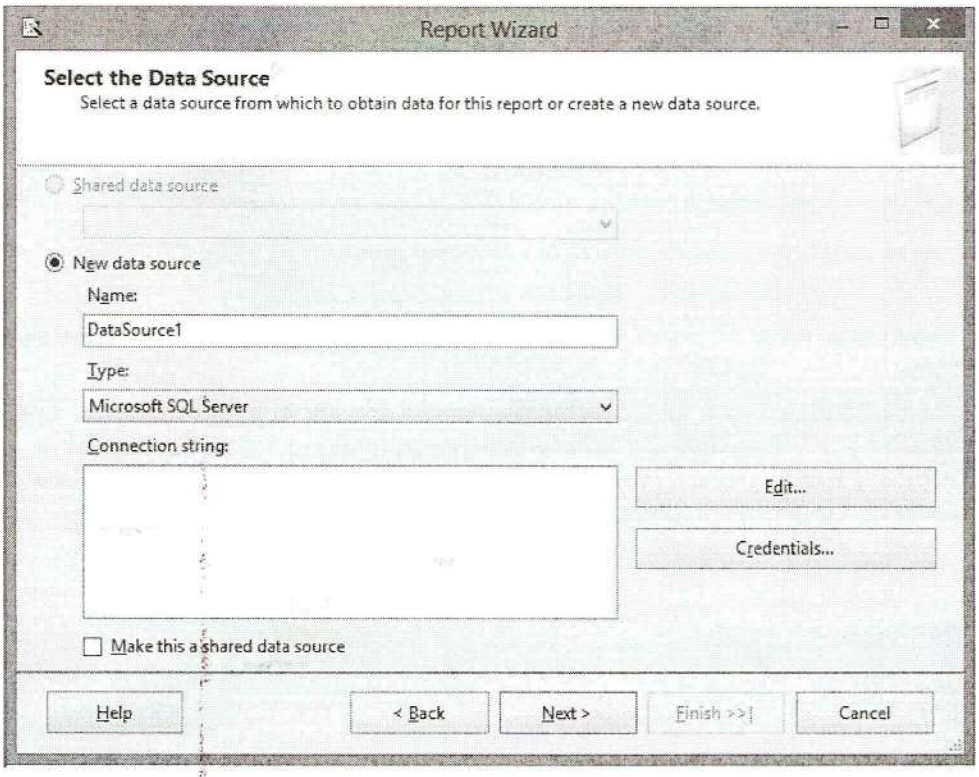
12.1.1. Crearea rapoartelor

Pentru crearea unui raport, se procedează în următorul mod:

1. Se lansează ***SQL Server Data Tools for Visual Studio 2013***.
2. În consecinţă, apare fereastra ***Visual Studio Starning Page***, în care se alege opţiunea ***New: Project...***
3. Apare caseta de editare ***New Project.*** Pe panelul ***Recent*** al acestei ferestre, se alege ramura ***Templates*** şi se alege tipul ***Business Intelligence***, apoi în panelul din dreapta se face un click pe şablonul ***Report Server Progect Wizard,*** în câmpul de editare ***Name:*** se specifică numele proiectului, iar în câmpul ***Location*** - locaţia fizică a fişierelor.



4. După aplicarea unui click pe butonul **OK**, va fi prezentată pe ecran fereastra ***Welcome to the Report Wizard*** cu un mesaj de bun venit al asistentului în cauză. În această fereastră, este suficient să se acționeze butonul **Next**.
5. Va fi deschisă fereastra ***Select the Data Source***, în care va trebui să se comunice sursa de date pentru crearea raportului. Aici există câteva opțiuni:
   - ***Shared data source.*** Se selectează pentru utilizarea unei surse predefinite de date partajate care, deja, conține toate informațiile de conectare la sursă. Lista conține toate sursele de date partajate care sunt incluse în proiect.



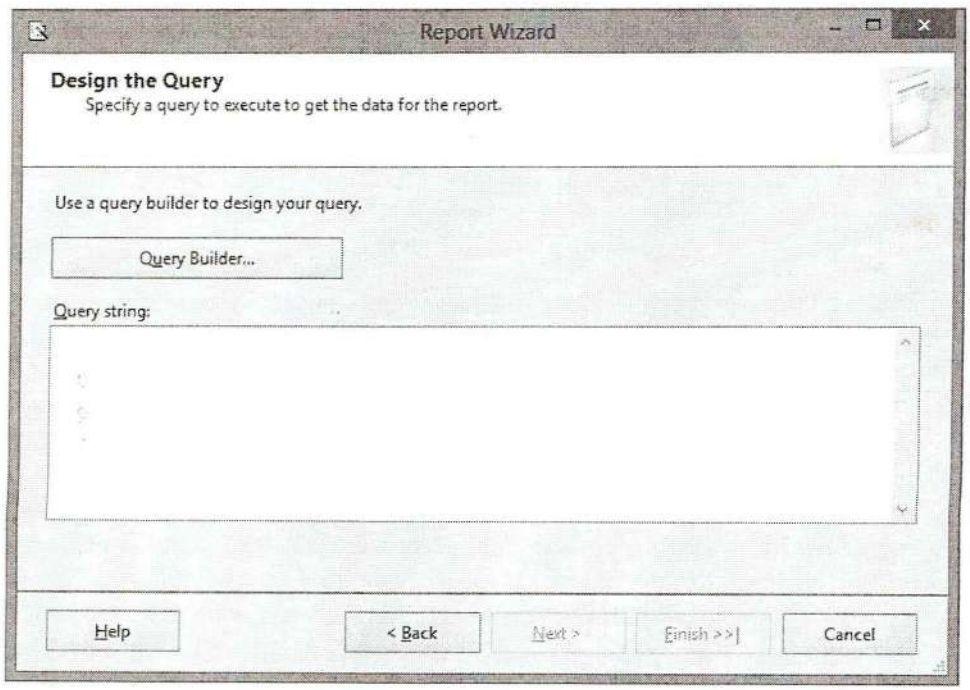
   - ***New data source.*** Se selectează pentru definirea unei surse noi de date. Informația despre sursă va fi utilizată doar cu raportul curent.

- ***Name.*** Se inserează numele sursei de date. Numele trebuie să fie unic în limitele raportului.
- ***Type***. Se alege tipul datelor, adică sistemul (ca în cazul nostru, SQL Server) sau limbajul (de exemplu, XML) în care aceste date au fost create.
- ***Connection string.*** Se selectează tipul secvenței de conectare a datelor-sursă. Pentru completarea proprietăților opțiunii ***Connection string,*** se apasă butonul **Edit...**

6. Apare, deja, fereastra cunoscută ***Connection properties***. Aici se specifică ***Server name*** și numele bazei de date (în cazul nostru, se alege baza de date ***calculatoare***). Apoi se apasă butonul **Test connection** pentru a verifica corectitudinea setărilor de conexiune, apoi se apasă **OK**. În urma setării proprietăților de conexiune, în câmpul ***Connection string*** apare informația despre sursa de date.

   Trebuie menționat că interfața ferestrei ***Connection properties*** depinde de tipul de conexiune ales. Pentru a continua, se apasă butonul **Next**.

7. La următorul pas al asistentului se deschide fereastra ***Design the Query***. Se utilizează pentru a crea o interogare sau în mod manual, sau cu ajutorul generatorului de interogări (***Query Builder***) pentru a construi interactiv interogarea, sau prin importarea interogării din alt raport.



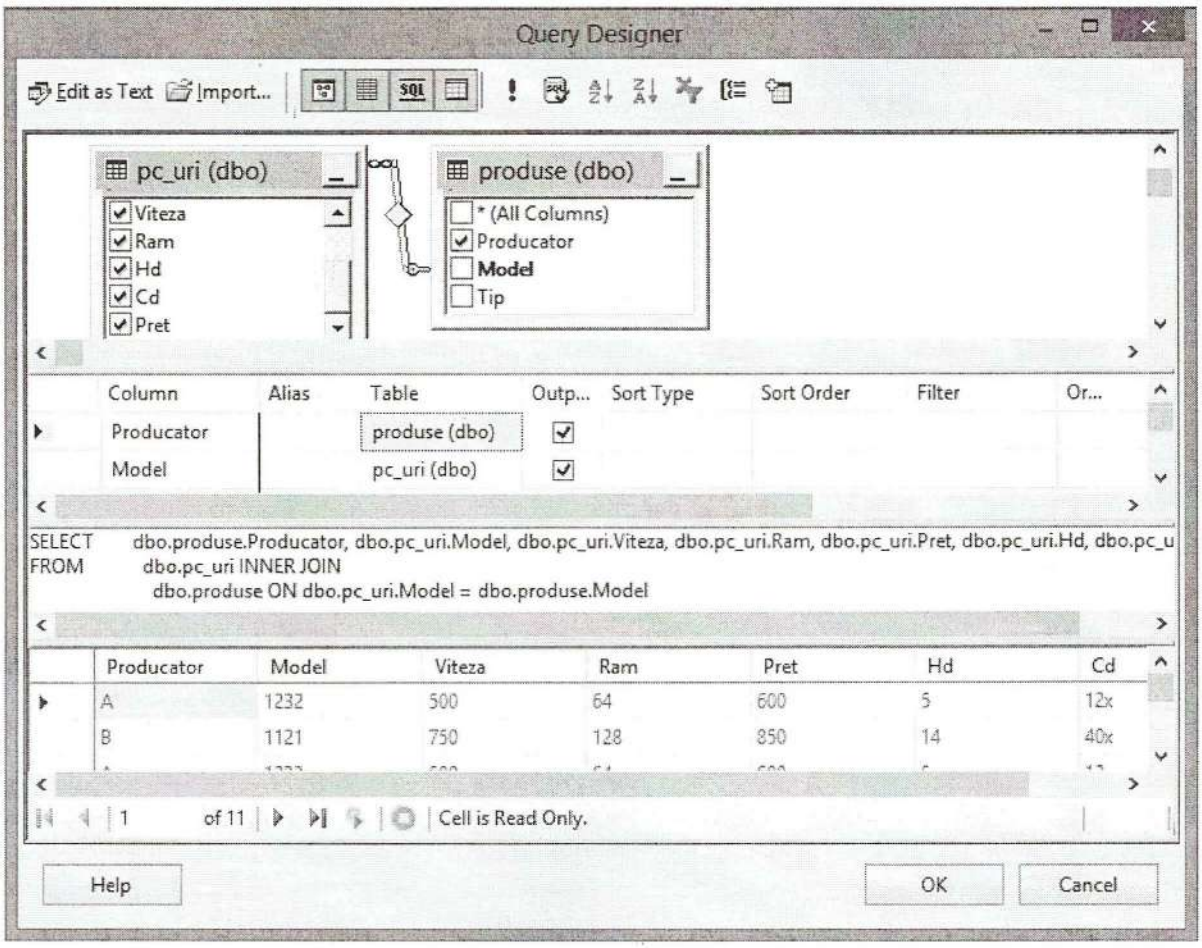
Sursa de date aleasă în pagina precedentă ***Select the Data Source*** a asistentului determină interogarea care poate fi inserată în această pagină.

De exemplu, în cazul în care sursa de date este de tip ***Microsoft SQL Server***, există posibilitatea introducerii instrucțiunii Transact-SQL sau numele procedurii stocate. În cazul în care sursa de date este de tip ***SQL Server Analysis Services***, panoul de interogări este dezactivat și nu există posibilitatea introducerii directe a interogării.

Această fereastră are două opțiuni:

- În câmpul de editare ***Query string,*** se scrie interogarea care caută datele ce trebuie folosite în raport.
- Butonul **Query Builder** deschide generatorul de interogări pentru sursa de date sau oferă posibilitatea de importare a interogării din alt raport.

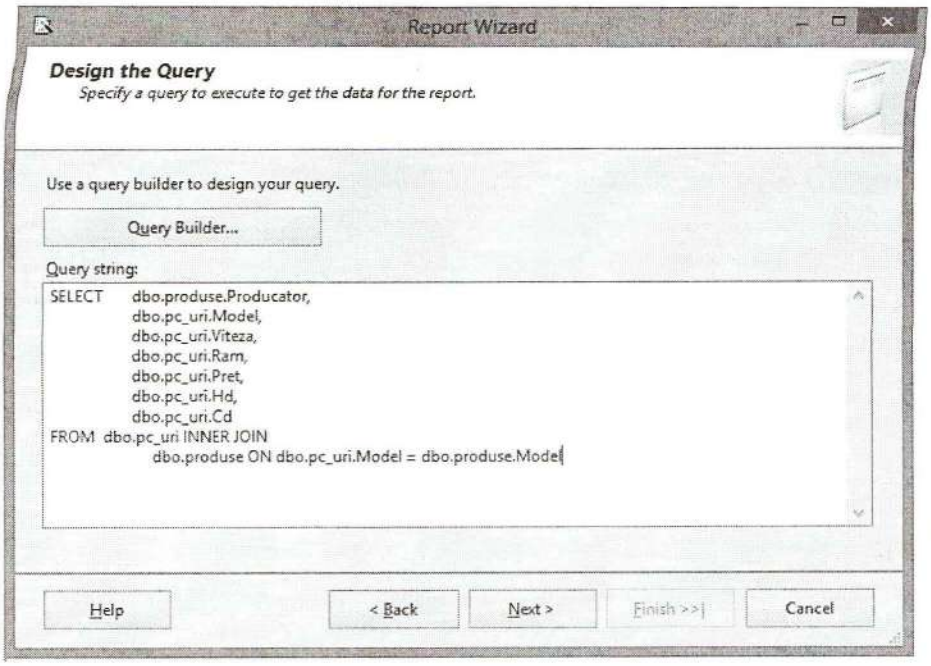
8. În cazul nostru, se alege opțiunea ***Query Builder...*** pentru a crea o interogare. Apare fereastra *constructorului de interogări,* unde se scrie o interogare al cărei rezultat va fi utilizat în raport.



9. Este posibilă alegerea uneia din cele două interfețe ale constructorului de interogări:
   - Constructorul de interogări bazat pe text (implicit) prevede o interfață de lucru simplă pentru specificarea interogării și vizualizarea rezultatelor. Este posibilă specificarea mai multor instrucțiuni Transact-SQL, interogări sau sintaxă a comenzilor de prelucrare a extensiilor de date, și interogări care sunt specificate drept expresii. Deoarece constructorul generic de interogări nu preprocesează interogarea și poate acomoda orice tip de sintaxă de interogări, el reprezintă instrumentul implicit de construire a interogărilor pentru ***Report Designer***.
   - Constructorul de interogări grafic oferă o interfață vizuală. Este utilizată în ***Visual Studio*** și în alte componente ale sistemului *SQL Server*. Se utilizează în cazul în care nu este nevoie de crearea expresiilor sau instrucțiunilor Transact-SQL complexe.

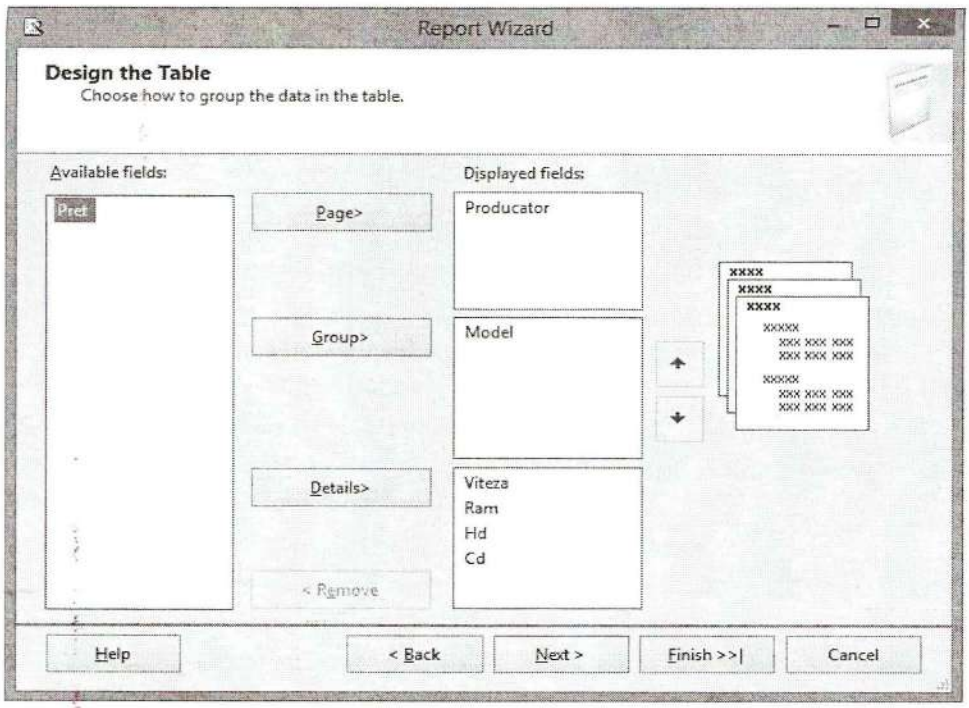
   Constructorul de interogări include opțiunile:
   - Pentru a accesa interfața grafică a constructorului, se dezactivează butonul **Edit As Text,** care se află pe bara de instrumente din colțul stâng-sus al ferestrei.
   - Butonul **Import** de pe bara de instrumente deschide caseta de dialog ***Import Query*** care afișează fișierele ***.rdl*** și ***.sql*** pentru orice raport disponibil. Interogarea importată poate fi utilizată așa cum este, sau poate fi modificată în constructor.
   - Câmpul **Command type** cuprinde următoarele tipuri de setări: text, procedură stocată sau tabel direct. Disponibilitatea tipului de comandă depinde de extensia datelor procesate.

   Restul butoanelor de pe bara de instrumente și panele din câmpul de lucru sunt similare celor din prezentate în capitolul.7, *Administrarea viziunilor*.
10. În cazul nostru, se dezactivează butonul **Edit As Text** și în interfața vizuală a constructorului, sunt specificate datele necesare pentru crearea raportului și se apasă butonul **OK**.



11. În consecință, în câmpul de editare ***Query string*** al ferestrei ***Design the Query*** a asistentului, apare interogarea construită în ***Query Builder.*** Apoi se apasă butonul **Next**.
12. În fereastra ***Select the Report Type*** apărută, se alege tipul raportului care trebuie creat. Se oferă două opțiuni: tipul ***Tabular*** sau tipul ***Matrix***.

    Fie se selectează tipul ***Tabular*** şi se apasă butonul **Next**.



13. Fereastra ***Design the Table*** include următoarele secțiuni:
    - ***Available fields.*** Se aleg câmpurile ce vor apărea în pagină mai sus de tabel, în calitate de câmp de grupare sau în detaliile tabelului. Pentru a muta un câmp din boxa ***Available*** în boxa ***Page, Group*** sau ***Details***, se selectează câmpul şi apoi se aplică un click pe butonul **Page, Group** sau **Details**. În mod alternativ, câmpul poate fi glisat în boxa respectivă.
    - ***Displayed fields.*** Afişează câmpurile, care vor fi reprezentate în boxele ***Page, Group*** sau ***Details***.

Fereastra ***Design the Table*** include următoarele butoane:

- **Page.** Afişează lista câmpurilor care vor apărea la nivel de pagină. Pentru a schimba ordinea câmpurilor, se selectează câmpul şi se aplică un click pe butonul sau butonul .

  La nivel de pagină, se va alege câmpul ***Producător***.
- **Group.** Afişează lista câmpurilor care grupează datele în tabel. Pentru a schimba ordinea câmpurilor, se selectează câmpul şi se aplică un click pe butonul sau butonul .

  Datele în raportul nostru vor fi grupate după modelul produsului, adică câmpul ***Model***.
- **Details.** Afişează lista câmpurilor care apar în secţiunea detaliilor tabelului. Pentru a schimba ordinea câmpurilor, se selectează câmpul şi se aplică un click pe butonul sau butonul .

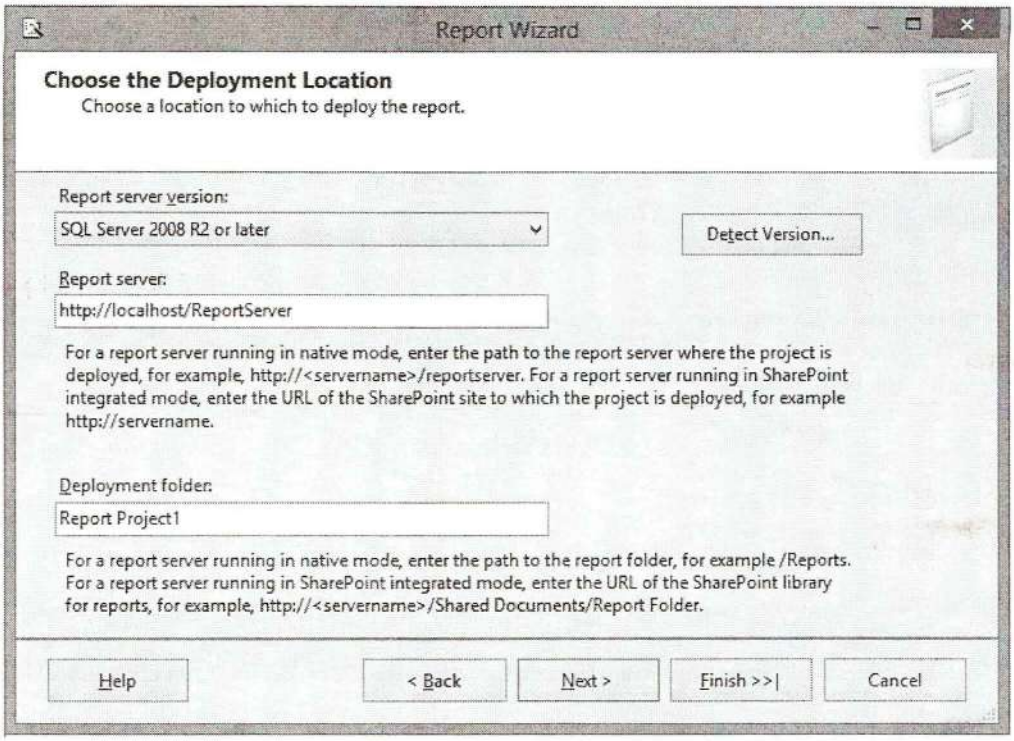
  În boxa ***Details***, se deplasează restul câmpurilor disponibile în secţiunea ***Available fields***.
- Pentru a şterge câmpul din listă se aplică un click pe butonul **Remove**.

După efectuarea setărilor respective, se apasă butonul **Next**.

14. În pagina ***Choose the Table Layout***, sunt câteva opţiuni:
    - ***Stepped.*** Se creează un raport ce conţine câte o coloană pentru fiecare câmp, cu câmpurile grupante amplasate în antetul grupurilor în partea stângă a coloanelor grupate, cu date detaliate. Acest tip de tabel nu conţine subsoluri grupate.
    - ***Block.*** Se creează un raport ce conţine câte o coloană pentru fiecare câmp, cu câmpurile grupante amplasate în primul rând detaliat al fiecărui grup. Acest tip de tabel are subsoluri grupate numai în cazul în care este selectată opţiunea ***Include subtotals.***
    - ***Include subtotals.*** Se selectează această opţiune atunci când se doreşte includerea în raport a subtotalurilor pentru câmpurile numerice. Dacă e selectată opţiunea ***Stepped***, atunci subtotalurile sunt plasate în rândurile-antet ale grupurilor. Dacă e selectată opţiunea ***Block***, atunci subtotalurile sunt plasate în rândurile-subsol ale grupurilor.
    - ***Enable drilldown.*** Se selectează această opţiune pentru a ascunde grupurile interne ale raportului, şi activarea funcţiei de vizualizare, ce rezultă într-un raport ***drilldown***.

    În cazul nostru, se lasă opţiunile implicite şi se apasă butonul **Next**.
15. Pagina ***Choose the Table Style*** se utilizează pentru selectarea stilurilor textelor şi liniilor din raport. Lista de stiluri se bazează pe un model configurabil.

    În cazul nostru, se lasă şablonul implicit ***Slate*** şi se apasă butonul **Next**.



16. Fereastra ***Choose the Deployment Location*** se foloseşte pentru setarea serverului de derulare şi a mapei pe server, unde se va păstra proiectul raportului. Această pagină oferă o serie de opţiuni:
    - ***Report server version.*** Se alege versiunea serviciului SSRS unde va fi executat raportul creat.
    - Butonul **Detect Version,** din partea dreaptă, analizează versiunea serviciului SSRS şi populează, în mod automat, câmpul ***Report server version***.
    - ***Report server.*** Se indică URL-ul directoriului virtual al serverului de derulare a rapoartelor, de exemplu, http://localhost/reportserver. Aceasta reprezintă calea virtuală către serverul de derulare a rapoartelor şi nu către *Gestionarul de Rapoarte* (*Report Manager*). Implicit, serverul de rapoarte e instalat cu un directoriu virtual ***reportserver***.
    - ***Deployment folder.*** Se indică mapa destinată publicării rapoartelor care se conţin în proiectul raportului de pe server. Valoarea implicită a mapei reprezintă numele proiectului. Dacă mapa nu există pe server, atunci ***Report Designer*** creează mapa respectivă în momentul publicării rapoartelor.

    Se lasă opţiunile implicite şi se apasă butonul **Next**.
17. Fereastră ***Completing the Wizard*** se foloseşte pentru atribuirea unui nume raportului şi revizuirea opţiunilor aplicate acestuia.
    - Numele raportului se specifică în câmpul ***Report name.***
    - Câmpul ***Report summary*** permite vizualizarea opţiunilor ce au fost selectate la pasul precedent al asistentului ***Report Wizard.***
    - Pentru modificarea unei opţiuni, se aplică un click pe butonul **Back** până la găsirea paginii cu opţiunea respectivă.
    - Se bifează radiobutonul ***Preview report*** pentru vizualizarea raportului după creare.

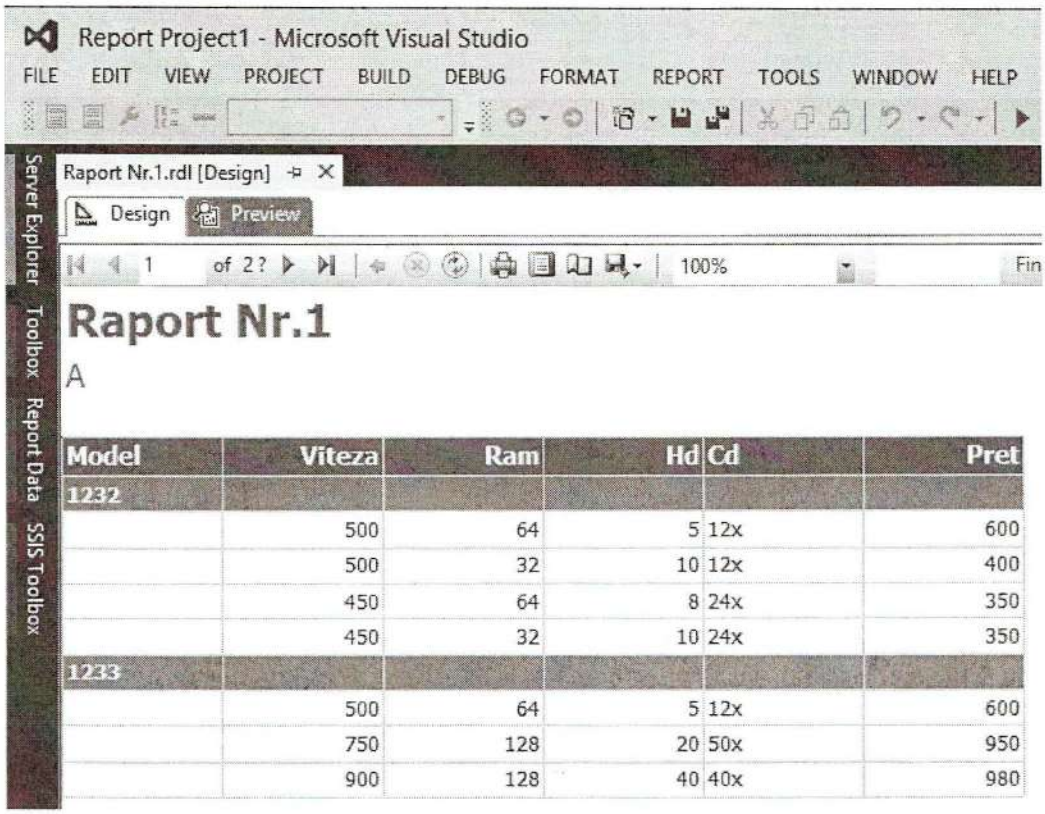
    Se apasă butonul **Finish**.

12.1.2. Previzualizarea rapoartelor

1. În consecinţă, se deschide fila ***Preview*** a constructorului de rapoarte. Pentru navigare se utilizează săgeţile **Next Page, Last Page** etc., de pe bara de instrumente.

   După cum se vede, în exemplul nostru, s-a obţinut un raport care include caracteristica

tehnică a calculatoarelor din tabelul ***pc_uri*** (în raport nu este inclus câmpul ***Cod***) şi datele despre producătorii acestor pc_uri (tabelul ***produse***, coloana ***Producator***). Raportul este grupat pe pagini după ***Producător***, datele de pe fiecare pagină, la rândul său, sunt grupate după ***Model***.



Raport Nr.1

A

Model	Viteza	Ram	Hd	Cd	Pret
1232					
	500	64	5	12x	600
	500	32	10	12x	400
	450	64	8	24x	350
	450	32	10	24x	350
1233					
	500	64	5	12x	600
	750	128	20	50x	950
	900	128	40	40x	980

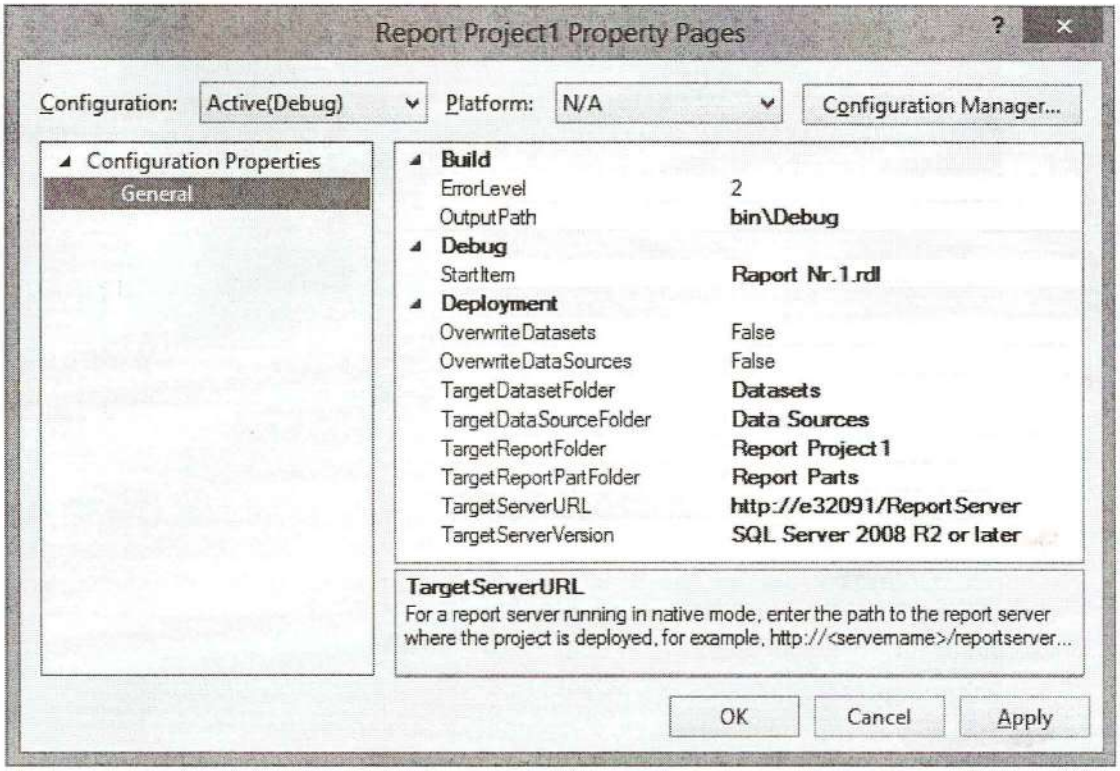
2. Pentru editarea proprietăților raportului deja creat, se activează fila ***Design.*** Trebuie menționat că pentru proiectarea eficientă a rapoartelor poate fi utilizat şi regimul ***design***.
3. După proiectarea şi testarea raportului acesta poate fi publicat pe serverul de lucru, utilizând instrumentele constructorului de rapoarte. În constructorul de rapoarte al mediului ***SQL Server Data Tools*** termenii *publicare* (***publish***) şi *desfăşurare* (***deploy***) sunt interschimbabili.

12.1.3. Configurarea proprietăților rapoartelor

1. Înainte de publicare, este important să fie corect configurate proprietățile de desfăşurare (***deploy***) a raportului.

   Pentru configurarea proprietăților unui proiect ***Report Server***, pe bara de meniuri se alege ***Project.*** În meniul apărut se selectează opțiunea ***Report <Nume de proiect> Properties.***

   În consecință, apare fereastra de dialog ***Report <Nume de proiect> Property Pages***.



Pe bara de instrumente în meniul derulant ***Configuration*** se alege una din cele trei configurații disponibile: ***Debug***, ***DebugLocal*** sau ***Release***.

- În mod implicit, opțiunea ***DebugLocal*** afișează raportul în fereastra locală de examinare prealabilă. Opțiunile ***Debug*** și ***Release*** publică raportul pe un server de rapoarte.
- Configurația activă este afișată în câmpul vizibil și este precedată de cuvântul ***Activ***. Există posibilitatea de vizualizare a mai multor configurații în același timp, selectând ***All Configurations*** sau ***Multiple Configurations***.

2. Pentru modificarea sau crearea unor configurații adiționale se apasă butonul **Configuration Manager** care cheamă fereastra de dialog respectivă. Orice modificare a configurării se reflectă în câmpul ***Configuration.***
   - În ***Project Contexts***, pentru fiecare proiect se bifează sau debifează opțiunea ***Build*** sau ***Deploy***.
   - Dacă este selectată opțiunea ***Build***, constructorul de rapoarte creează un proiect și verifică erorile înainte de previzualizare sau de publicare pe serverul de rapoarte.
   - Dacă, însă, este selectată opțiunea ***Deploy***, constructorul de rapoarte publică raportul pe serverul de rapoarte, conform proprietăților de desfășurare definite.
   - Dacă ***Deploy*** nu este selectat, constructorul de rapoarte afișează raportul specificat în proprietatea ***StartItem*** într-o fereastră locală de previzionare.

   Secțiunea de proprietăți ***Bulid***
   - ***ErrorLevel*** specifică gravitatea problemelor de generare semnalate ca erori. Problemele cu niveluri de gravitate mai mică sau egală cu valoarea ***ErrorLevel*** sunt semnalate ca erori, în caz contrar, sunt semnalate ca avertizări. Orice eroare va provoca eșecul sarcinii de generare. Nivelurile de gravitate sunt cuprinse între 0 și 4 inclusiv. Valoarea implicită este 2.
   - ***OutputPath*** indică calea de stocare a definiției raportului, care este utilizată la verificarea *build*-ului, *deployment*-ului și vizualizării rapoartelor. Calea trebuie să fie diferită de calea utilizată pentru proiect.

Secțiunea de proprietăți ***Debug***

- În câmpul ***StartItem*** se alege raportul care este afișat în navigatorul web după ce proiectul este publicat pe serverul de rapoarte sau pe pagina de previzualizare în cazul rulării locale a proiectului. ***StartItem*** este necesar pentru configurări care execută dar nu desfășoară proiectul și pentru utilizarea comenzii ***Debug*** **(F5)**. Este obligatoriu pentru configurările care execută desfășurarea proiectului.

Secțiunea de proprietăți ***Deployment***

- ***OverwriteDatasets*** indica daca definițiile seturilor de date pot fi suprascrise în timpul desfășurării proiectului.
- În câmpul ***OverwriteDataSources*** se selectează valoarea ***True*** pentru suprascrierea sursei de date de pe server cu sursa de date a proiectului, atunci când rapoartele sunt publicate. Se alege ***False***, pentru păstrarea sursei de date existente pe server.
- ***TargetDatasetFolder*** indică calea spre dosarul disponibil setului de date.
- În câmpul ***TargetDataSourceFolder*** se scrie numele dosarului în care se păstrează sursele de date partajate publicate. Dacă astfel de dosar nu este specificat, sursa de date este publicată în același directoriu ca și raportul. În cazul în care dosarul nu există pe serverul de rapoarte, ***Report Designer*** creează dosarul atunci când rapoartele sunt publicate.
- ***TargetReportFolder*** indică numele dosarului în care se păstrează rapoarte publicate. În mod implicit, numele dosarului e reprezentat de numele proiectului. În cazul în care dosarul nu există pe serverul de rapoarte, ***Report Designer*** creează dosarul atunci când rapoartele sunt publicate.
- ***TargetReportPartFolder*** indică calea către dosarul componentelor raportului.
- ***TargetServerURL*** indică adresa URL a serverului de rapoarte. Înainte de publicarea unui raport, este important să fie setată o adresă URL validă.

  În cazul în care are loc publicarea pe un server de rapoarte care rulează în mod nativ, se utilizează URL-ul directoriului virtual al serverului de rapoarte. De exemplu, *http:// <server> / ReportServer*. Acesta este directorul virtual al serverului de rapoarte, dar nu al ***Report Manager*-ului**. În mod implicit, serverul de rapoarte este instalat într-un directoriu virtual numit "***ReportServer***".

  Atunci când are loc publicarea pe un server de rapoarte care rulează în modul de integrare ***SharePoint***, se utilizează o adresă URL de cel mai înalt nivel a unui sit ***SharePoint*** sau unui subsit. Dacă situl nu este specificat, este folosit situl de nivel superior implicit. De exemplu, *http:// <numeserver>*, *http:// <numeServer / <sit>* sau *http:// <numeserver> / <sit> / <subsit>*.
- ***TargetServerVersion*** această opțiune este descrisă în cadrul procedurii de creare a raportului, folosind asistentul.

3. În cazul nostru, proprietățile se configurează așa cum sunt prezentate în imaginea de mai sus. Proprietățile implicite nu au fost modificate substanțial.
4. În caseta de dialog ***Configuration Manager*** au fost bifate ambele opțiuni: ***Build*** și ***Deploy*** pentru a depista erorile posibile încă la etapa de construire înainte de a porni desfășurarea.
5. În câmpul de editare ***TargetServerURL*** este definită calea corectă către serverul de rapoarte local (*http://e32091/ReportServer*).
6. După toate setările necesare, se apasă **OK**.

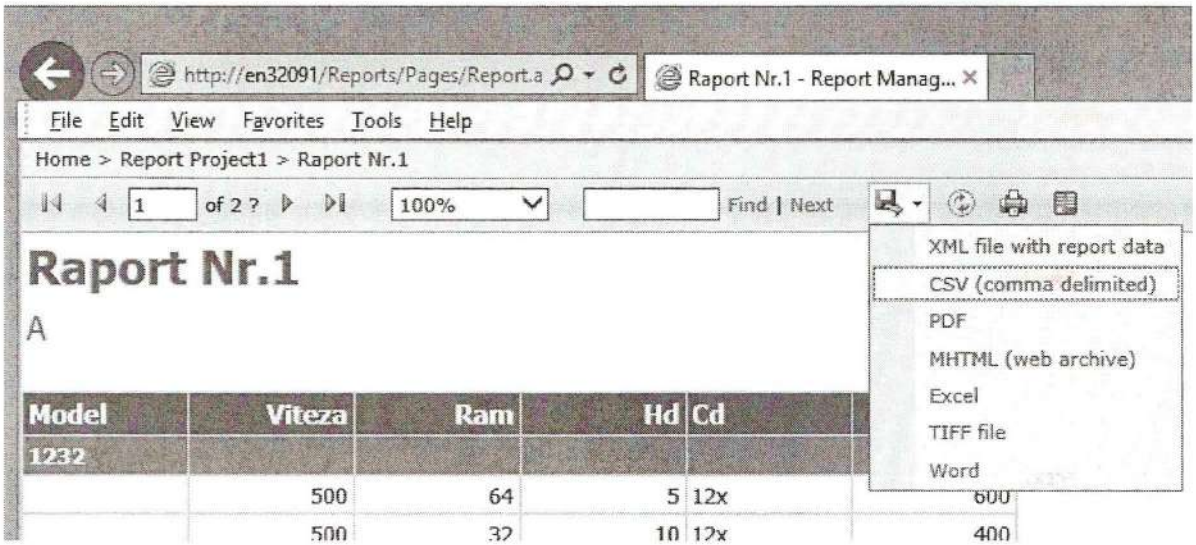
12.1.4. Publicarea și vizualizarea rapoartelor

1. Apoi în fereastra din dreaptă - ***Solution Explorer***, se alege raportul necesar din arborele de obiecte, se face un click-drept pe acesta și în meniul de context se alege **Start**.

   Statutul procesului de publicare se afișează în fereastra ***Output***. Dacă totul se desfășoară normal, în rezultat trebuie să se deschidă în mod automat o fereastră a navigatorului web cu raportul creat.

Raportul creat poate fi deschis într-un navigator în orice moment. Pentru aceasta, pe bara de adrese a navigatorului ***Internet Explorer,*** se specifică adresa serverului de rapoarte, unde se află raportul creat.

2 În cazul nostru, adresa serverului este *http://e32091/Reports*. În pagina apărută, se alege proiectul în cauză - ***Report Project1*** şi apare raportul respectiv sub forma unei pagini ***asp***.



Pagina WEB apărută este dotată cu o bară de instrumente ce permit:

- listarea paginilor raportului;
- exportarea datelor în diferite formate (de exemplu, XML, PDF, CSV etc.);
- imprimarea raportului.

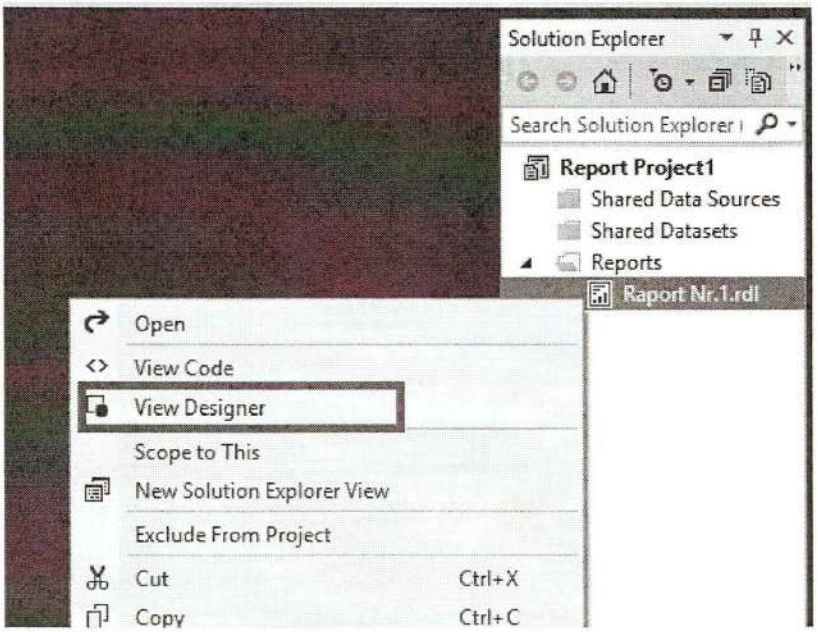
3. Pentru a exporta datele, din meniul drop-down care apare după acționarea butonului ***Export drop down meniu,*** se alege tipul necesar de reprezentare a raportului. Datele exportate sunt vizualizate în mediul aplicațiilor respective, de unde pot fi uşor salvate pe un suport fizic.

Trebuie menționat că, actualmente, există o serie de instrumente pentru publicarea bazei de date în WEB şi de fiecare dată sunt tot mai simple.

12.2. Editarea proprietăților unui raport deja existent

Pentru editarea proprietăților unui raport deja creat, se parcurge următoarea cale:

1. Se lansează ***SQL Server Data Tools***.
2. În fereastra ***Visual Studio Starning Page*** se alege opțiunea ***Open: Project...***
3. În caseta de dialog ***Open Project*** apărută se specifică calea spre ***Report Project*** necesară şi se apasă butonul **OK**.
4. În consecință, apare interfața constructorului de rapoarte.
5. Pentru a edita un raport dintr-un proiect, în fereastra ***Solution Explorer*** în arborele de obiecte se alege raportul respectiv şi se face un click-drept pe acesta.
6. În meniul de context apărut se alege opțiunea ***View Designer*** sau în cazul utilizatorilor mai experimentați ***View Code***. Ca rezultat, apare fila respectivă a constructorului.



Regimul de constructor, bineînțeles, oferă o flexibilitate cu mult mai mare în alegerea formelor de publicare a datelor. In afara de o serie de instrumente grafice oferite de ***Visual Studio*** şi ***Report Builder***, pentru a mari flexibilitatea rapoartelor, utilizatorul poate folosi expresii dezvoltate în limbajul ***Visual Basic***. De asemenea, intru-un proiect SSRS, pot fi uşor incluse şi clase scrise in *C#*. Acest fapt face posibilitățile de design a interfețelor rapoartelor SSRS aproape nelimitate.

Pentru publicarea rapoartelor este important ca serviciul de rapoarte să fie pornit. Proprietățile serviciilor de rapoarte pot fi setate în mediul SQL Server 2014. Procedura de conectare la ***Reporting Services*** în mediul ***SQL Server Management Studio*** este similară cu conectarea la alte servicii din interfața SQL Server şi a fost analizată în capitolele precedente.

Întrebări de control:

1. Particularitățile procesului de creare şi gestiune a rapoartelor cu ajutorul asistentului şi în regim de constructor.
2. Setările serviciilor de rapoarte necesare pentru construirea şi desfăşurarea rapoartelor.

Sarcini practice:

1. Să se creeze un fişier HTML şi să se publice în acesta rezultatul executării procedurii stocate, create în capitolul 8, sarcina practică 1.
2. Să se creeze pe masa de lucru un fişier HTML şi să se scrie în acesta înregistrările din tabelul ***pc_uri*** (baza de date ***calculatoare***), care vor reprezenta procesoarele ce au o frecvență de tact mai mare de 500 MHz şi prețul mai mic de 900. Să se afişeze numai primele 2 înregistrări.
3. În baza exercițiilor rezolvate în capitolul 4, să se creeze 2 rapoarte. Primul - cu ajutorul asistentului, al doilea - în regim de designer. Datele pentru primul raport să se obțină în ***Query Builder*** şi să fie vizualizat sub formă tabelară în formatul ***asp***. Datele pentru al doilea raport să reprezinte rezultatul executării interogării scrise direct în ***Query string*** şi să fie vizualizat sub formă matricială într-un fişier ***pdf***.

13. SERVICII DE ANALIZĂ DIMENSIONALĂ A DATELOR

Capitolul este consacrat caracterizării instrumentelor de prelucrare a datelor multidimensionale, fiind prezentate metodele de creare şi gestionare a cuburilor *n*-dimensionale, utilizând componenta ***Microsoft SQL Server 2014 Analysis Services***. O deosebită atenţie este acordată proiectării şi menţinerii cuburilor OLAP (*Online analytical processing*), revizualizării şi modificării structurii cubului, încărcării datelor într-un cub, desfăşurării şi procesării cubului, precum şi interogării cubului [1,5].

Analysis Services reprezintă soluţia OLAP cu care Microsoft a pătruns pe piaţa ***Business Intelligence***. OLAP este o metodă de acces rapidă şi comodă, de vizualizare şi analiză a informaţiilor de afaceri care este realizată printr-un model de date, intuitiv clar, organizat în formă de cub *n*-dimensional. ***Microsoft SQL Server 2014 Analysis Services*** (*SSAS*) furnizează instrumente pentru elaborarea, proiectarea şi menţinerea cuburilor şi a altor obiecte OLAP susţinute.

13.1. Introducere în OLAP

Un cub este o structură multidimensională ce conţine dimensiuni şi măsuri (variabile). Dimensiunile definesc structura cubului, iar măsurile furnizează utilizatorului valori numerice care caracterizează obiectul economic. Valorile, „amplasate" de-a lungul dimensiunilor, se numesc membri. În calitate de structură logică, cubul permite unei aplicaţii-client să obţină datele în aşa mod, de parcă celulele cubului definesc orice valoare agregată posibilă. Poziţia celulelor în cub sunt definite de intersecţia dimensiuni-membru. Dimensiunile se folosesc în alegerea şi agregarea datelor la nivelul de detaliere cerut. Ele se organizează în ierarhii, constituite din mai multe nivele, fiecare din ele reprezentând un nivel de detaliere, necesar pentru o analiză dată.

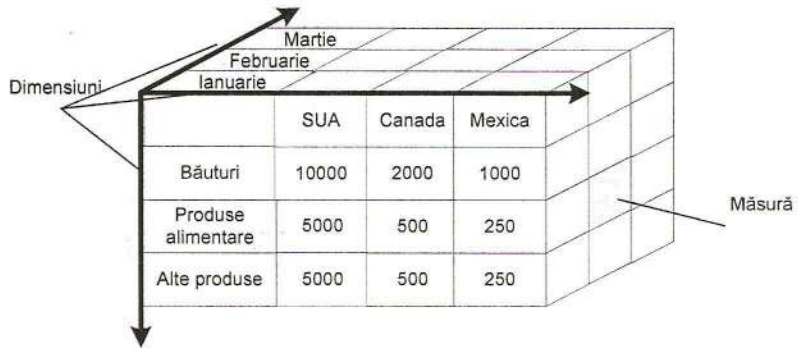
În ***Microsoft SQL Server 2014 Analysis Services***, un cub este elaborat în baza tabelelor şi viziunilor. Măsurile cubului reprezintă fapte, iar dimensiunile lui sunt domenii de interes, cum ar fi timpul, produsele şi consumatorii. Un cub multidimensional este creat, folosind calcule, KPI-uri (*key performance indicators*), acţiuni, partiţii, perspective şi interpretări. Conceptul de cub este sinonim conceptului UDM (*Unified Dimensional Model*), care scoate hotarul dintre modelul relaţional şi cel multidimensional.

Măsurile cubului se bazează pe coloanele din unul sau mai multe tabele-fapte şi membrii dimensiunilor cubului sunt bazate pe tabele-dimensiuni. Un cub poate fi elaborat şi fără sursa de date relaţională de bază. În acest caz, structura relaţională de bază poate fi generată pentru a susţine cubul. Faptele în cub sunt agregate în baza ierarhiei dimensiunilor. Dimensiunile sunt bazate pe atributele care sunt puse în corespondenţă cu una sau mai multe coloane din tabelele-dimensiuni, iar ierarhiile se definesc în baza acestor atribute.

13.1.1. Un exemplu de cub

Cubul permite stocarea datelor agregate şi ierarhice. Un cub este format din mai multe *dimensiuni*, iar agregarea datelor se face după atribute denumite *măsuri*. O dimensiune poate avea mai multe *nivele*.

În calitate de măsuri, în cubul tridimensional din figura de mai jos, sunt folosite volumele vânzărilor, iar ca dimensiuni – timpul, produsul şi magazinul. Dimensiunile sunt reprezentate pe anumite nivele de grupare: produsele sunt grupate după categorii, magazinele – pe ţări, iar datele despre timpul efectuării operaţiei – pe luni. Mai jos, vor fi analizate mai detaliat nivelele de grupare (ierarhii).



13.1.2. "Tăierea" cubului

Chiar şi cubul tridimensional e greu de afişat pe ecran, în aşa fel, ca să se vadă valorile măsurilor dezirabile. Dar ce să mai vorbim despre cuburile cu un număr de dimensiuni mai mare decât trei? Pentru vizualizarea datelor păstrate în cub, se folosesc, de obicei, tabelele bidimensionale obişnuite cu denumiri ierarhice de rânduri şi coloane.

Vizualizările bidimensionale ale cubului pot fi obținute, prin „tăierea" lui de-a curmezişul uneia sau mai multor axe (dimensiuni): se fixează valorile tuturor dimensiunilor, în afară de două şi se obține un tabel obişnuit. Pe axa orizontală a tabelului (denumirile coloanelor) e prezentată o dimensiune, pe cea verticală (denumirile rândurilor) – alta şi în celulele tabelului – valoarea măsurilor. Măsurile sunt, practic, considerate drept una din dimensiuni: sau se alege vizualizarea unei măsuri (atunci coloanele şi rândurile sunt două dimensiuni), sau se vizualizează câteva măsuri (şi atunci pe o axă a tabelului vor fi denumirea măsurilor, iar pe alta - valorile unicei dimensiuni „netăiate").

În figura următoare, e prezentată tăietura cubului pentru o măsură – *Unit Sales* (bucăți vândute) şi două dimensiuni „netăiate" – *Store* (Magazin) şi *Time* (Timpul).

	SUA	*Canada*	*Mexica*
Ianuarie	20 000	4000	2000
Februarie	30 000	6000	3000
Martie	50 000	10000	5000

În următoarea figură e prezentată doar o dimensiune „netăiată" – *Store*, însă, sunt valorile câtorva măsuri – *Unit Sales* (bucăți vândute), *Store Sales* (volumul vânzărilor) şi *Store Cost* (cheltuielile)

	SUA	*Canada*	*Mexic*
Unit Sales	2000	400	200
Store Sales	30 000	6000	3000
Store Cost	10 000	2000	1000

Vizualizarea bidimensională a cubului e posibilă şi atunci când rămân „netăiate" şi mai mult de două dimensiuni. Astfel, pe axele tăieturii (rânduri şi coloane) pot fi amplasate două sau mai multe dimensiuni ale cubului „tăiat", precum în următoarea figură:

	Ianuarie			*Februarie*		
	SUA	*Canada*	*Mexic*	*SUA*	*Canada*	*Mexic*
Unit Sales	500	100	50	500	100	50
Store Sales	7500	1500	750	7500	1500	750
Store Cost	2500	500	250	2500	500	250

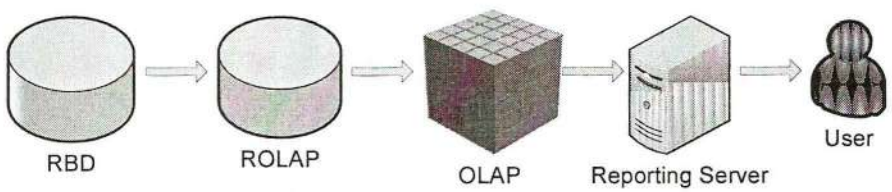
13.1.3. Metode de construire a cuburilor

Precum s-a menționat anterior, utilizarea bazelor de date multidimensionale (OLAP) reprezintă o metodă de acces rapidă şi comodă de vizualizare şi analiză a informației.

Principalul avantaj în folosirea cuburilor este oferit de răspunsul rapid la cererile complexe datorită modului de organizare a datelor pe dimensiuni şi măsuri, permițând şi agregarea acestora. Dacă bazele de date relaționale sunt orientate pentru păstrarea datelor în forma normalizată, compactă şi consistentă ce oferă o viteza relativ mare în procesarea instrucțiunilor DML, bazele de date OLAP, datorită formei denormalizate de păstrare a datelor, sunt orientate spre executarea extrem de rapidă a instrucțiunilor de consultare. De asemenea, bazele de date OLAP păstrează şi date sub forma precalculată fapt ce sporeşte viteza de afişare a datelor agregate.

Bazele de date OLAP, de regula, se construiesc având ca sursă de date bazele de date relaționale. Datele din bazele de date relaționale prin metode ETL (*Extract - Transform - Load*) sunt transformate în baze de date OLAP.

Se recomandă ca în procesul de importare a datelor din bazele de date relaționale native (RBD) în bazele de date OLAP şa fie creată o baza de date analitică relațională (ROLAP) care, pe de o parte, să comunice cu RDB şi, pe de altă parte, cu baza de date OLAP. Baza de date ROLAP reprezintă o baza de date relațională, alcătuită din datele dezagregate care sunt aranjate în tabele-masuri şi tabele-dimensiuni. Baza de date ROLAP ca şi alte baze relaționale este gestionată de ***Database Engine Service***. Astfel, procesul ETL se va petrece ca în următoarea figură:



O astfel de arhitectură a procesului de analiză şi calcul oferă o flexibilitate şi independență mai mare fiecărui nivel de procesare. De asemenea, prelucrarea datelor în cadrul procesului ETL, pe fiecare nivel, va decurge mai repede şi cu costuri hardware mai mici.

Instrumentele ***SQL Server Data Tools*** (SSDT) acceptă, în cadrul unui proiect *Analysis Services*, două metode de utilizare a schemelor relaționale în definirea obiectelor OLAP.

În general, în cadrul unui proiect *Analysis Services*, obiectele OLAP se definesc pe baza unui model de date logic construit într-o viziune a sursei de date, numită DSV, (*Data Source View*) a unui proiect. Această viziune a sursei de date este definită pe baza unor elemente ale schemei care provin din una sau mai multe surse de date relaționale.

Dar se poate începe, de asemenea, cu definirea obiectelor OLAP şi, ulterior, să se genereze o viziune a sursei de date (DSV), o sursă de date, precum şi schema bazei de date relaționale (ROLAP) care suportă aceste obiecte OLAP. Această bază de date relațională se numeşte "zonă care reprezintă subiectul bazei de date".

Această abordare este, uneori, numită metoda de proiectare top-down şi este frecvent folosită pentru crearea prototipurilor şi modelor de analiză. În continuare, va fi aplicată această abordare.

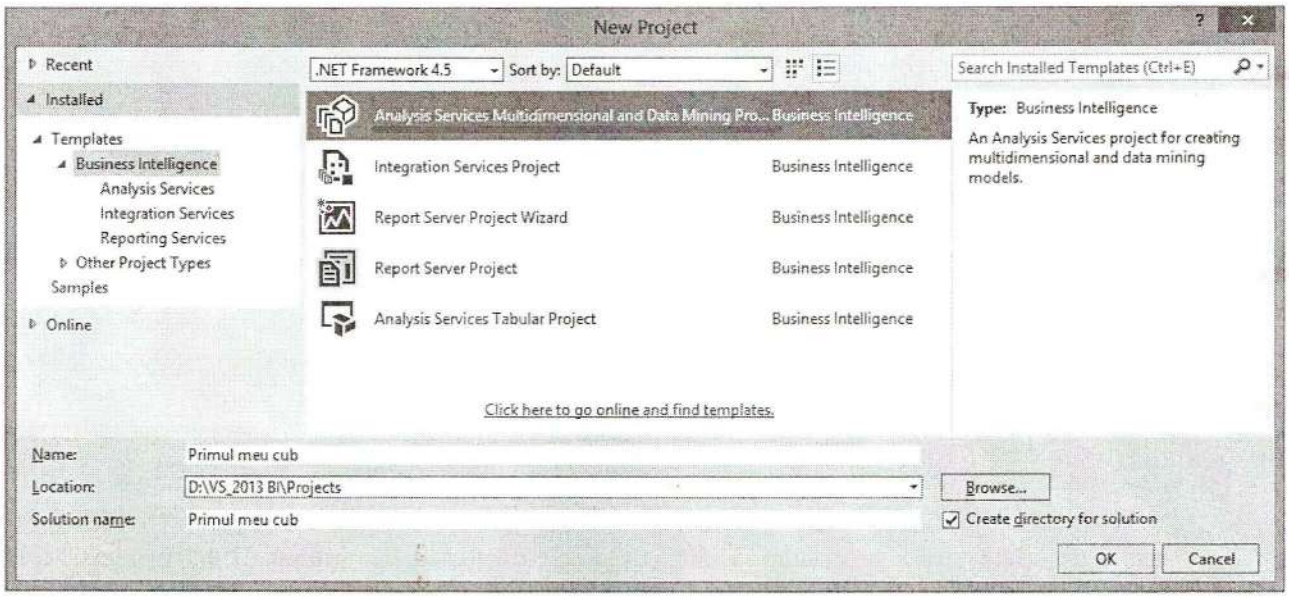
13.2. Crearea unui proiect Analysis Services

În ***SQL Server Data Tools for Visual Studio***, înainte de construirea cubului, este necesar să se creeze un proiect ***Analysis Services*** în care va fi păstrat cubul. Un proiect ***Analysis Service*** creează o bază de date OLAP care poate include în sine mai multe cuburi, dar orice cub nu poate exista în afara unui proiect.

Următoarea procedură va crea un proiect ***Analysis Services,*** numit ***Primul meu cub,*** pentru a stoca primul cub.

1. Se lansează SQL Server Data Tools for Visual Studio 2013.
2. În fereastra ***Start Page*** se alege opțiunea ***New Project...***

   În rezultat, apare fereastra de dialog ***New Project*** care afișează instrumente de modelare accesibile pentru proiecte ***Business Intelligence*** (BI). Dacă mediul de dezvoltare ***Visual Studio*** este instalat pe calculator, e posibilă, prin alte instrumente de modelare, adăugarea la proiect a altor componente, dacă e necesară integrarea BI cu alte aplicații.
3. Se alege tipul de proiect Business Intelligence, apoi se face un click pe șablonul Analysis Services Multidimensional and Data Mining Project.
4. În câmpul de editare ***Name,*** se inserează numele proiectului ***Primul meu cub.***



5. Se aplică un click pe butonul **OK**.

   Fereastra ***Solution Explorer*** acum va poseda un set de dosare goale, care, mai târziu, vor conține obiectele pe măsură ce vor fi construite în proiectul ***Analysis Services.***

Îndată ce proiectul ***Analysis Services*** este creat, poate fi proiectat cubul cu sau fără tabele-fapte și tabele-dimensiuni gata de a fi utilizate în calitate de surse de date. Cubul constă dintr-una sau mai multe măsuri, cum ar fi ***Sales*** (Vânzări) sau ***Quantity*** (Volum) din tabelul-fapte și una sau mai multe dimensiuni, cum ar fi ***Product*** (Produs) sau ***Time*** (Timp) din tabelul-dimensiuni. Când se construiește cubul cu surse de date gata, asistentul ***Cube Wizard*** ajută la selectarea tabelelor-dimensiuni și tabelelor-fapte potrivite.

13.3. Crearea unui Cub folosind asistentul, dar fără sursa de date

Dacă se decide construirea cubului fără sursa de date de tip Warehouse, există opțiunea de utilizare a șabloanelor integrate care vor fi de ajutor la definirea structurii cubului. După selectarea șablonului dezirabil, se selectează măsurile și dimensiunile din șablon pentru a le include în cub și se specifică dimensiunile, care vor fi asociate cu măsuri. Există opțiunea de generare imediată a schemei relaționale după terminarea lucrului cu asistentul. Totuși, înainte de a face acest pas, se poate dori, mai întâi, modificarea sau suprimarea obiectelor create din șablon.

Urmează procedura de creare a unui cub nou cu ajutorul asistentului.

13.3.1. Construirea unui Cub nou

1. În ***Solution Explorer***, se aplică un click-drept pe dosarul ***Cubes*** și apoi se face un click pe opțiunea ***New Cube***. Se afișează pagina ***Welcome to the Cube Wizard*** a asistentului. Asistentul poate fi lansat și din meniul ***Project-> New Cube ...*** de pe bara de meniuri.

2. Se aplică un click pe butonul **Next**.

   În următoarea procedură va fi selectată opțiunea pentru crearea cubului cu formarea manuală a tabelelor.

13.3.2. Specificarea metodei de construire

Apare fereastra ***Select Creation Method*** care include câteva opțiuni de creare a cubului. Pentru crearea cubului, utilizând tabele existente, se bifează ***Use existing tables***. Pentru crearea cubului vid, se alege ***Create an empty cube***, iar pentru formarea manuală a tabelelor cubului, se recurge la opțiunea ***Generate tables in the data source.***.

1. Se selectează opțiunea ***Generate tables in the data source***.
2. În câmpul de editare ***Template***, se selectează dintr-o listă derulantă opțiunea ***Adventure Works Standard Edition***
3. Se aplică un click pe butonul **Next**.

   În funcție de opțiunea aleasă pe pagina ***Creation Method,*** la următorul pas al asistentului se va afișa pagina respectivă.

   Deoarece pe pagina ***Creation Method*** a fost selectată opțiunea ***Generate tables in the data source,*** apare pagina ***Define New Measures.***

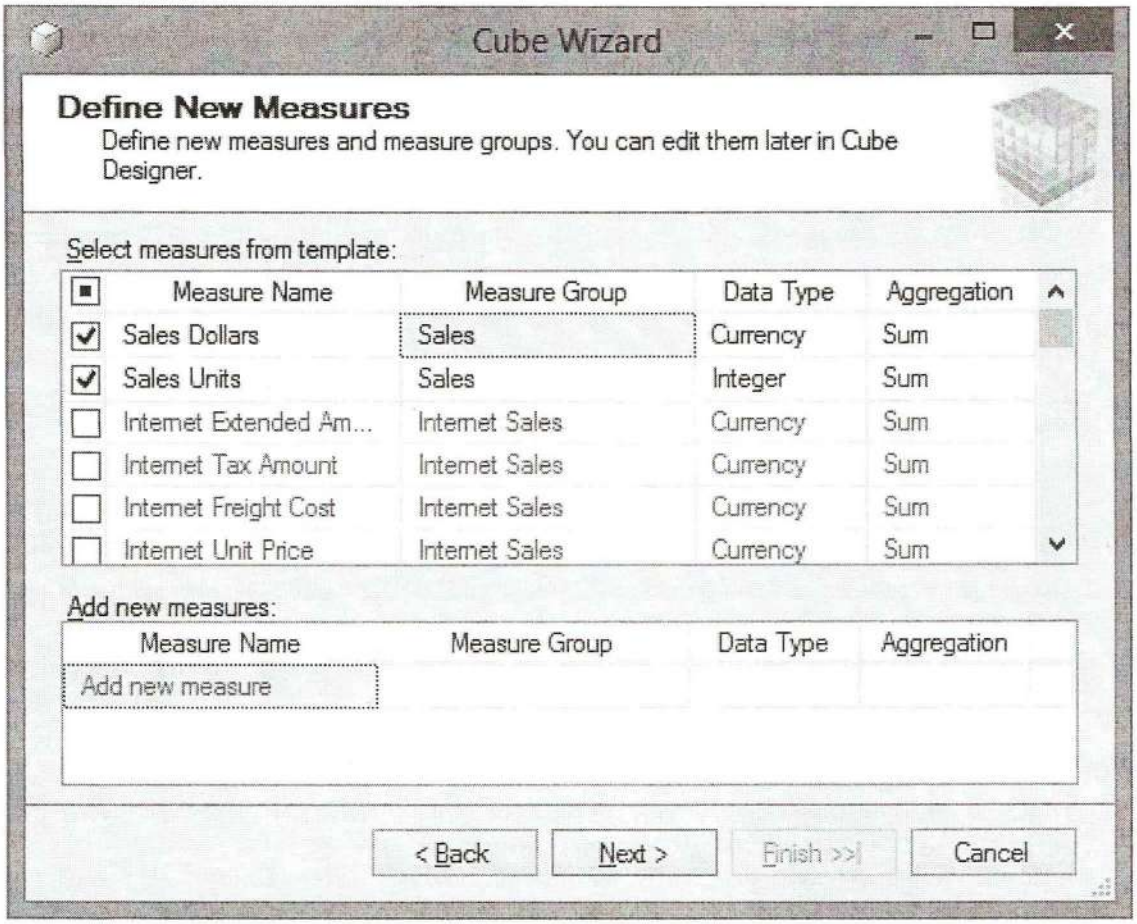
   În următoarea procedură, vor fi definite pentru cub două măsuri, ***Sales Amount*** (Suma vânzărilor) și ***Order Quantity*** (Cantitatea comandată).

13.3.3. Adăugarea măsurilor

1. Pentru a suprima selectările tuturor măsurilor, se face un click pe butonul cu puncte de suspensie din stânga butonului **Measure Name,** și apoi se bifează căsuțele de selecție necesare, adică căsuțele din stânga opțiunilor ***Internet Sales Amount*** și ***Internet Order Quantity***.

   Trebuie menționat că aceste măsuri sunt parte a unui grup de măsuri numit ***Internet Sales.*** Dacă se răsfoiește toată lista de măsuri, se pot observa alte grupuri de măsuri: ***Internet Orders, Internet Customers, Sales Reasons, Reseller Sales, Reseller Orders, Sales Summary, Sales Orders*** și ***Sales Targets***. Fiecare grup de măsuri este o colecție de măsuri ce provin din același tabel-fapte.
2. Se revine la începutul listei de măsuri, se aplică un click în boxa ce conține ***Internet Sales Amount*** și se schimbă numele măsurii în ***Sales Dollars***.
3. Se repetă pasul precedent pentru a schimba numele ***Internet Order Quantity*** în ***Sales Units***. Astfel, numele măsurilor oferite de șabloane pot fi schimbate, ca să satisfacă mai bine sensul. Dacă s-a uitat să se modifice numele acestui pas al asistentului ***Cube Wizard***, acest lucru poate fi făcut și în ***Cube Designer.***
4. În coloana ***Measure Group,*** pentru rândurile ***Sales Dollars*** și ***Sales Units,*** se modifică numele grupului de măsuri ***Internet Sales*** în ***Sales.***

   Pagina ***Define New Measures*** a asistentului, acum, va arăta asemenea celei de mai jos:



5. Se aplică un click pe butonul **Next**.

   În fereastra ***Define new Dimensions*** ce apare, la cubul creat se adaugă dimensiunile ***Product*** şi ***Date***.

13.3.4. Adăugarea dimensiunilor

1. Pentru ştergerea dimensiunilor oferite de şablon, se face un click pe butonul din stânga butonului **Type**.

   Şabloanele conţin o serie de dimensiuni care pot fi introduse în cub. Fiecare dimensiune, la rândul său, conţine atribute de stocare care nu pot fi modificate în limitele asistentului ***Cube Wizard***, dar ele pot fi liber modificate după terminarea lucrului cu asistentul.

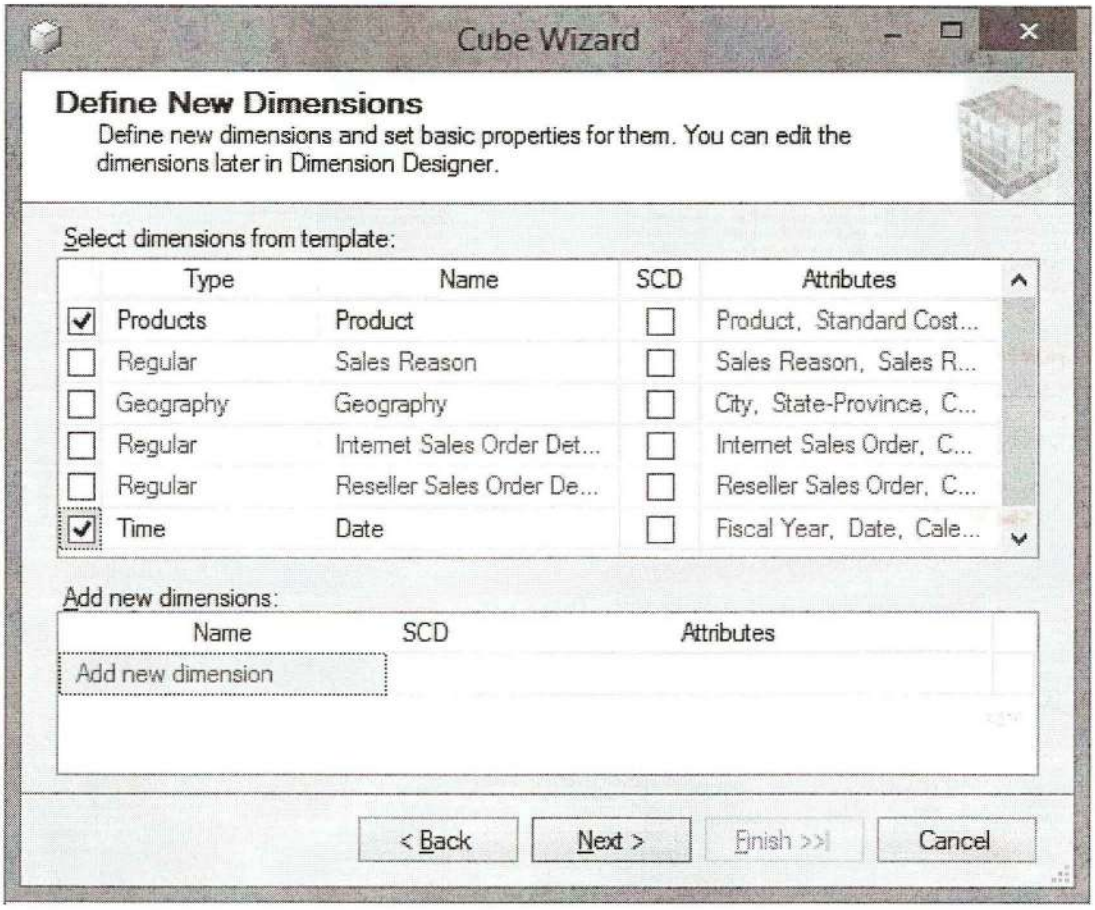
2. Se aplică un click (căsuţa din stânga) pe dimensiunile ***Products*** şi ***Time***.

   Ecranul va arăta în modul următor:

3. Să se considere atributele implicite pentru fiecare dimensiune. Întotdeauna există posibilitatea de suprimare a atributelor ne-necesare sau de adăugare a atributelor dezirabile după terminarea lucrului cu asistentul. Acum sunt prezente toate dimensiunile necesare pentru construirea cubului. Se poate trece la următorul pas.

4. Se aplică un click pe butonul **Next**.

   În următoarea fereastră ***Define Time Periods***, se specifică datele de început şi sfârşit ale datelor ce vor fi analizate. De exemplu, ianuarie 1, 2010 şi decembrie 31, 2014, respectiv.



13.3.5. Definirea perioadelor de timp

1. Se aplică un click în meniul derulant ***First Calendar Day.*** Se parcurge calendarul acționând săgeata din stânga până la luna iulie 2010, apoi se aplică un click pe data de început – 1.

   În cazul în care se foloseşte şablonul dimensiunii ***Time***, e necesară definirea domeniului de valori ale acestei dimensiuni. De asemenea, trebuie specificate acele atribute care vor fi incluse în dimensiunea ***Time***. De exemplu: ***an***, ***trimestru*** şi ***lună***.

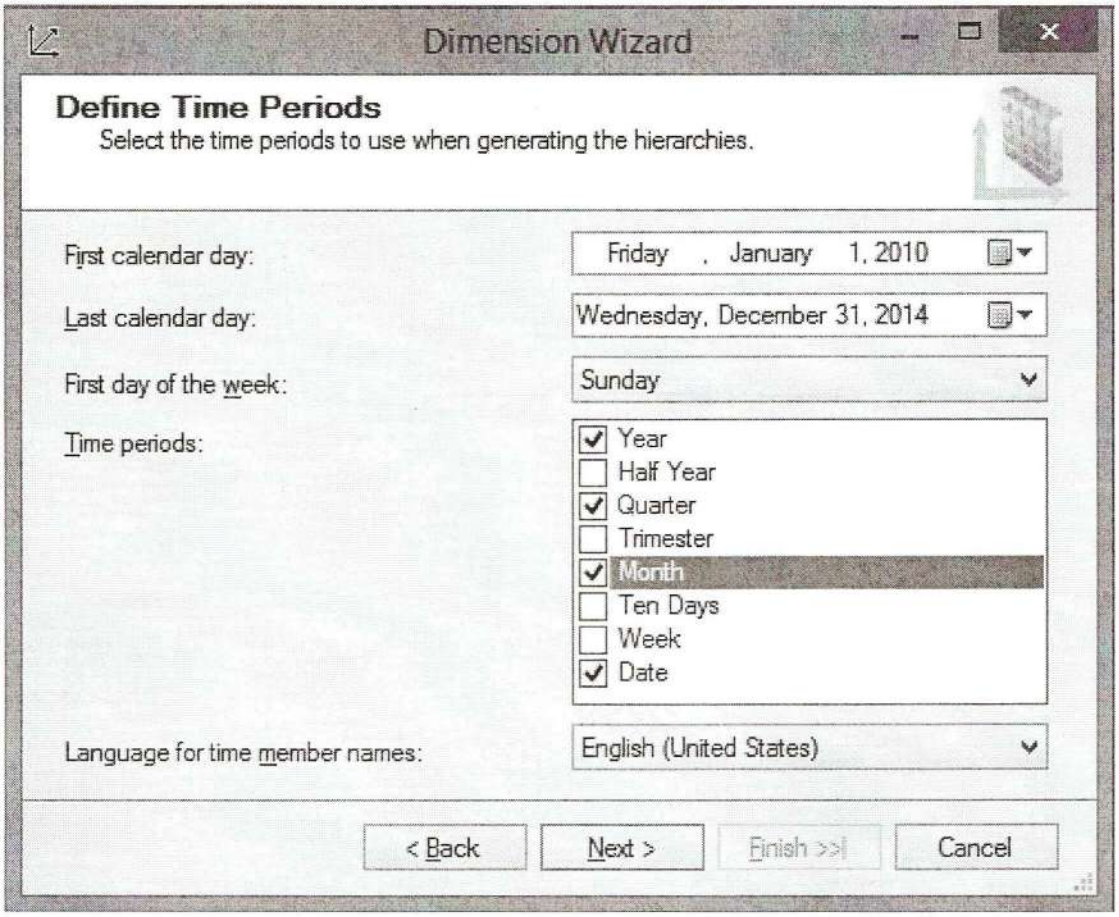
2. Se selectează 2014 din meniul derulant ***Last Calendar Day***.

   În câmpurile de editare valorile date pot fi inserate, dar poate fi folosită, pentru aceasta, şi componenta calendar.

3. În panoul ***Time Periods,*** sunt selectate ***Year***, ***Quarter*** şi ***Month***.

4. În panoul ***Time Periods,*** se scoate bifa din dreptul atributului ***Date***.

   Odată cu debifarea atributului **Date**, apare un mesaj de atenționare în partea de jos a boxei de dialog a asistentului: "*The date frequency level is required. If you do not want it to be visible, you can hide this attribute in the dimension editor.*" Din motivul că atributul **Date** al dimensiunii **Time** este unul cheie, el trebuie neapărat să fie inclus. În interiorul Analysis Services, se foloseşte un atribut-cheie pentru determinarea univocă a fiecărei înregistrări. Trebuie menționat că trecerea, la următoarea pagină, nu poate fi realizată fără a fi inclus atributul **Date**.



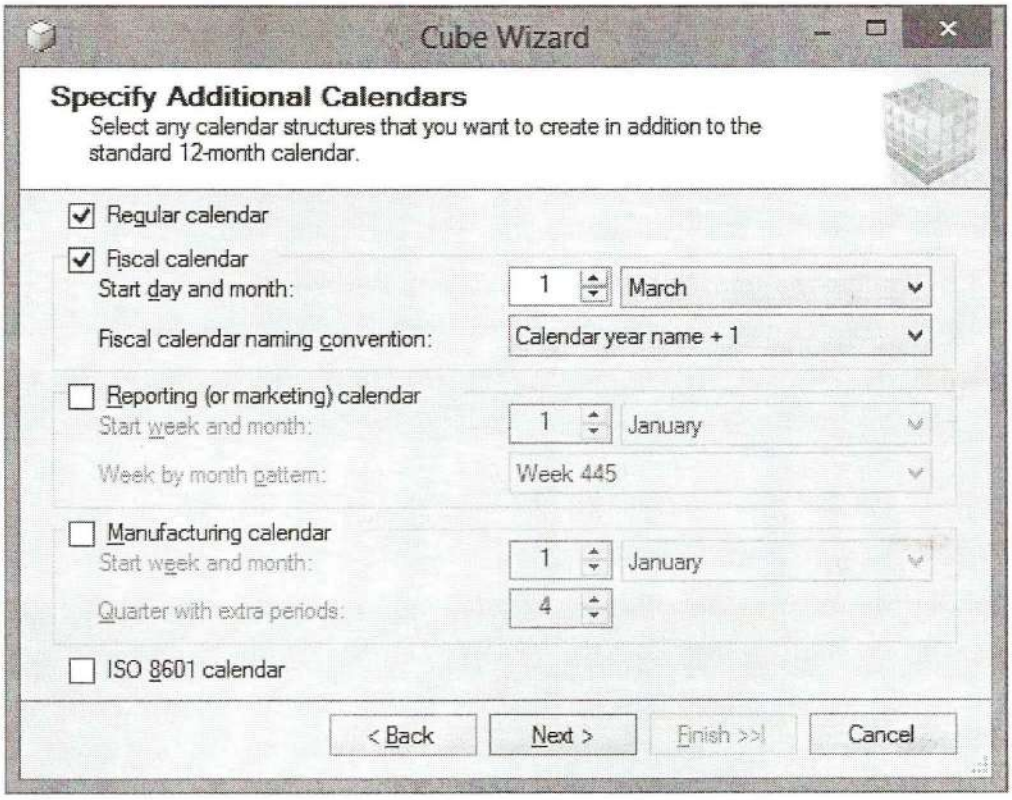
5. Se bifează atributul ***Date***, şi se apasă butonul **Next**.

   În fereastra ***Specify Additional Calendars*** se va adăuga un calendar fiscal, care începe cu *March 1*.

13.3.6. Adăugarea calendarului special

1. În pagina apărută, se bifează căsuța ***Fiscal Calendar***.

   Calendarele adiționale sunt opționale. Pentru crearea ierarhiilor alternative ale atributului ***Time,*** sunt disponibile patru tipuri de calendare: ***Fiscal***, ***Reporting***, ***Manufacturing*** şi ***ISO 8601*** (*International Standards Organization*). În cazul utilizării tuturor calendarelor (cu excepția ***ISO 8601***), pentru alocarea potrivită a zilelor într-o perioadă de timp, poate fi indicată data de început. De exemplu, dacă se schimbă ziua de start în calendarul ***Fiscal Calendar*** în *1 March*, atunci, pe parcursul sesiunii de vizualizare a cubului, zilele lunii *March* vor fi afişate ca zile ale primei luni a anului fiscal.
2. Se setează **Start Day And Month** cu valoarea 1 March.

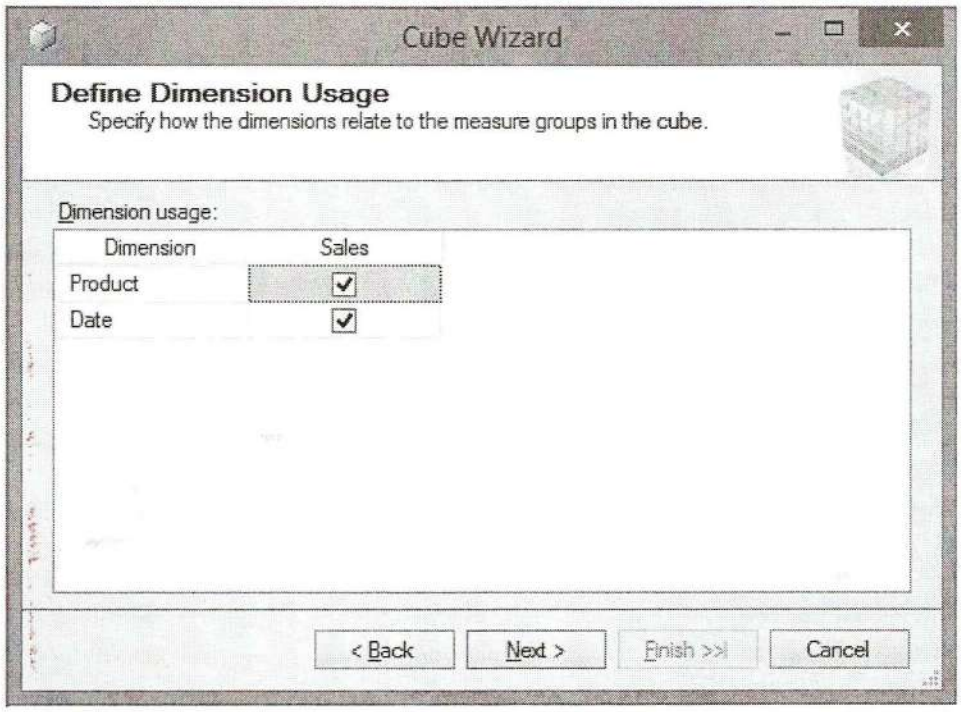


3. Se aplică un click pe butonul **Next**.

   În următoarea fereastră ***Define Dimension Usage***, se vor specifica legăturile dintre dimensiuni şi grupurile de măsuri.

13.3.7. Stabilirea legăturilor dintre dimensiuni şi măsuri

1. Se consideră următoarele setări implicite de pe pagina ***Define Dimension Usage*** a asistentului ***Cube Wizard***:



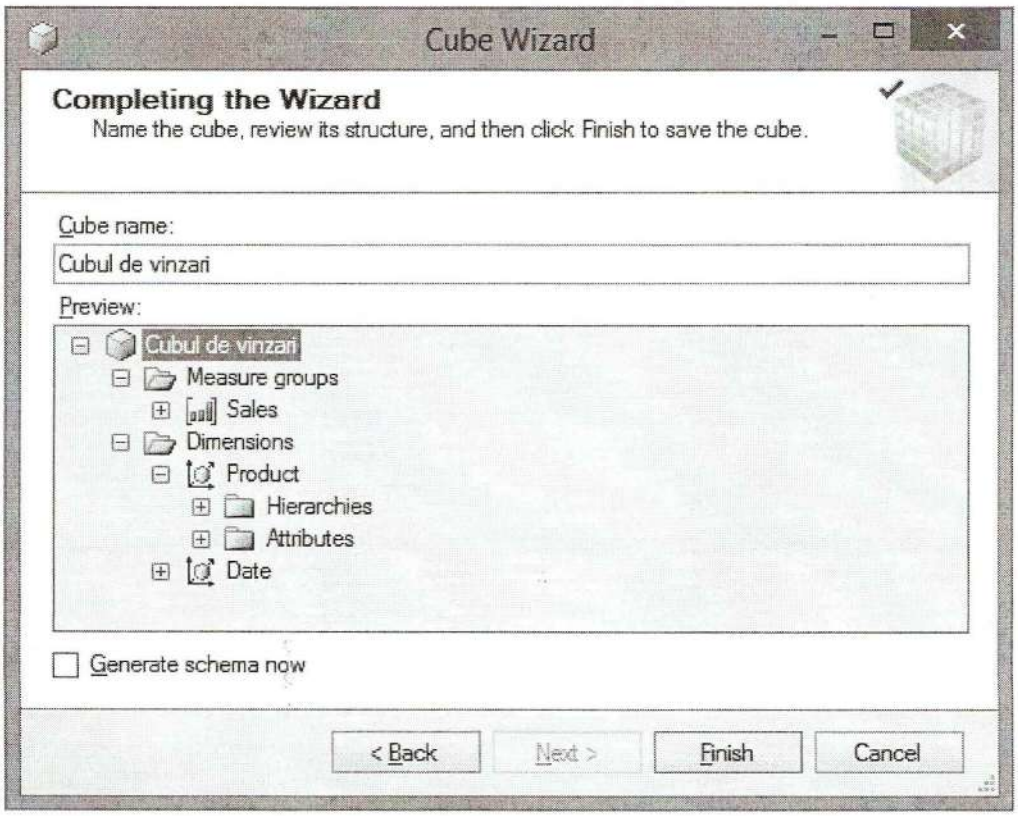
Căsuțele bifate indică faptul că ambele dimensiuni ***Product*** și ***Date*** vor fi asociate automat cu grupul de măsuri ***Sales***. Deoarece va fi construit un cub simplu, ambele dimensiuni vor fi asociate cu o singură măsură.

2. Se aplică un click pe butonul **Next**.

   În următoarea fereastră ***Completing the Wizard***, se oferă un nume cubului și constituie ultimul pas executat de asistent.

13.3.8. Finalizarea utilizării asistentului Cube Wizard

1. În câmpul de editare ***Cube Name***, se inserează numele: ***Cubul de vinzari***.



Cubului i se atribuie automat un nume, atunci când se folosește un șablon. În acest caz, numele implicit este ***Adventure Works***. Dacă în procesul de creare a cubului nu se utilizează un șablon, atunci numele implicit este ***New Cube***. Numele cubului poate fi oricând modificat în procesul de lucru.

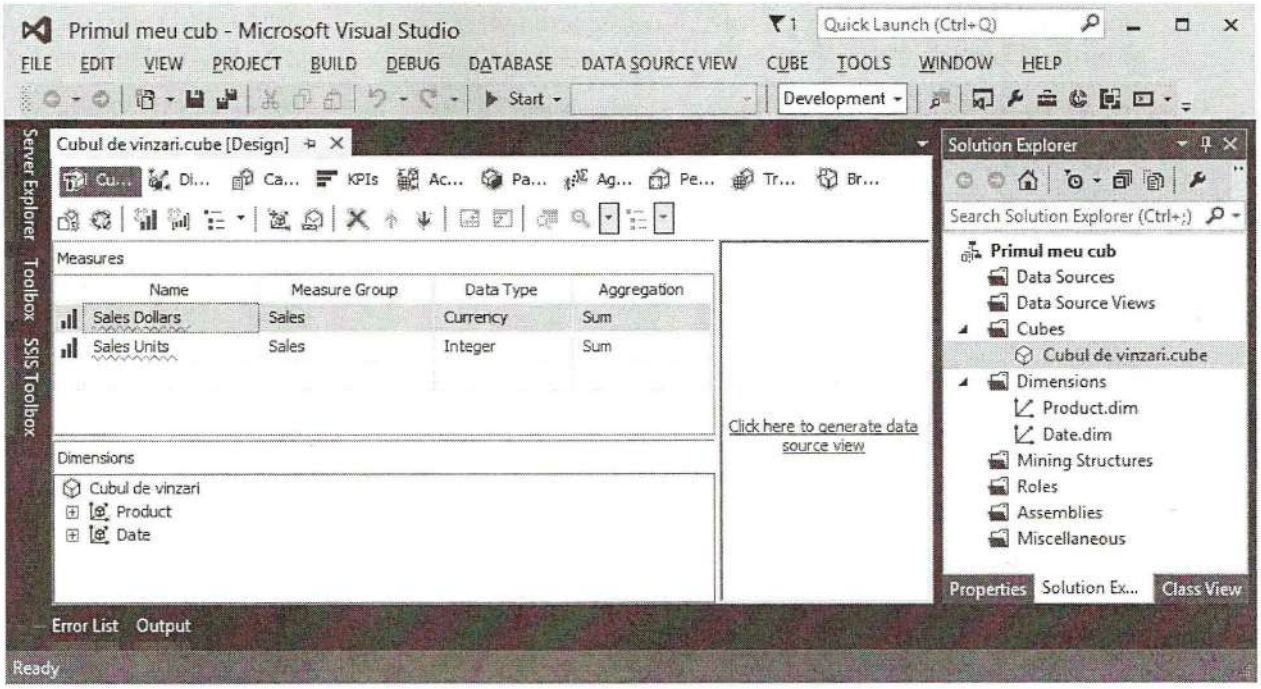
2. Structura cubului creat care este prezentată sub forma unui arbore ierarhic poate fi revăzută în câmpul de vizualizare ***Preview***. Se expandează nodul ***Product*** și apoi se expandează dosarul ***Attributes,*** pentru a vizualiza atributele care vor fi create în dimensiunea dată.

   Aceasta este prima vizualizare comprehensivă a structurii cubului. Pentru efectuarea modificărilor în structura cubului, este posibilă revenirea la pagina respectivă a asistentului ***Cube Wizard***. Cu toate acestea, modificarea structurii cubului poate fi realizată și ulterior, dacă se dorește adăugarea sau suprimarea obiectelor.

   Dacă se bifează radiobutonul de jos ***Generate Schema Now***, se va lansa asistentul ***Schema Generation Wizard***. Schema este o bază de date relațională denormalizată ce conține tabele pentru grupurile de măsuri și dimensiuni utilizate în cub. Acest asistent va fi examinat mai târziu.

3. Se acționează butonul **Finish**.

   În rezultat, apare fila ***Cube Structure*** a instrumentului ***Cube Designer***



Trebuie menționat că liniile roșii din panoul ***Measures*** indică faptul că există o problemă în ceea ce privește măsurile afișate. Deoarece nu este nici o viziune sursă de date (*DSV*), apare o eroare condiționată. Această eroare va fi înlăturată mai târziu la generarea schemei.

În colțul stânga-jos, se poate observa panoul ***Dimensions,*** care conține acele două dimensiuni ale cubului. În colțul stânga-sus, se poate observa panoul ***Measures,*** unde se poate vedea grupul de măsuri cu două componente. Deoarece cubul nu este încă conectat la sursa de date, nu există un *DSV* vizibil în centrul ferestrei. Cu toate acestea, structura cubului poate fi revăzută și modificată până aceasta devine corectă. Asistentul ***Cube Wizard*** nu creează fișierele cubului și nu încarcă valorile din magazia de date (*data warehouse*).

13.4. Revizualizarea structurii cubului în Cube Designer

Fila ***Cube Structure*** a instrumentului ***Cube Designer*** reprezintă obiectele-dimensiuni și măsuri care au fost definite pentru cub. În acest spațiu de lucru, pot fi revăzute și modificate proprietățile fiecărui obiect. Atunci când se selectează vreun obiect, proprietățile acestuia sunt disponibile pentru redactare în fereastra ***Properties*** (dacă fereastra ***Properties*** nu este afișată în câmpul de lucru, aceasta poate fi chemată apăsând tasta **F4**).

Fila ***Cube Structure*** oferă și un panel-diagramă în care se poate vizualiza structurile tabelelor de bază pentru cub cu scopul de a vedea coloanele din tabele-dimensiuni și tabele-fapte ce sunt asociate. Dacă aceste tabele nu sunt create, pe panel va apărea un link - ***Click Here To Generate Data Source View,*** la acționarea căruia se va lansa asistentul ***Schema Generation Wizard.***

În următoarea procedură se va seta în calitate de măsură implicită, măsura ***Sales Dollars*** și se va explora fila ***Cube Structure*** a instrumentului ***Cube Designer*** .

1. Se aplică un click pe ***Sales Dollars*** în fila ***Cube Structure***.

   Se consideră fereastra ***Properties*** ce apare mai jos de fereastra ***Solution Explorer.*** Ea reprezintă efectiv toate proprietățile disponibile pentru obiectul selectat, adică pentru ***Sales Dollars.***

2. Se acționează butonul ***Show Measures Tree,*** , de pe bara de instrumente.

   Panoul ***Measures*** afișează grupul de măsuri ***Sales*** și măsurile asociate în formă de arbore.

3. Se efectuează un click pe ***Sales*** în arborele măsurilor.

   În fereastra ***Properties,*** se pot vizualiza toate proprietățile asociate grupului ***Sales*** de măsuri. La vizualizarea măsurilor în formă de grid (*Grid View*), care reprezintă forma de vizualizare

implicită în ***Cube Structure***, nu pot fi accesate proprietățile grupurilor de măsuri. Totuși, poate fi utilizată bara de instrumente pentru a trece dintr-o formă de vizualizare în alta. Butonul de trecere de la ***Grid View*** la ***Tree View*** și viceversa este .

4. Se aplică un click pe cubul ***Cubul de vinzari*** de pe panelul ***Measures***.

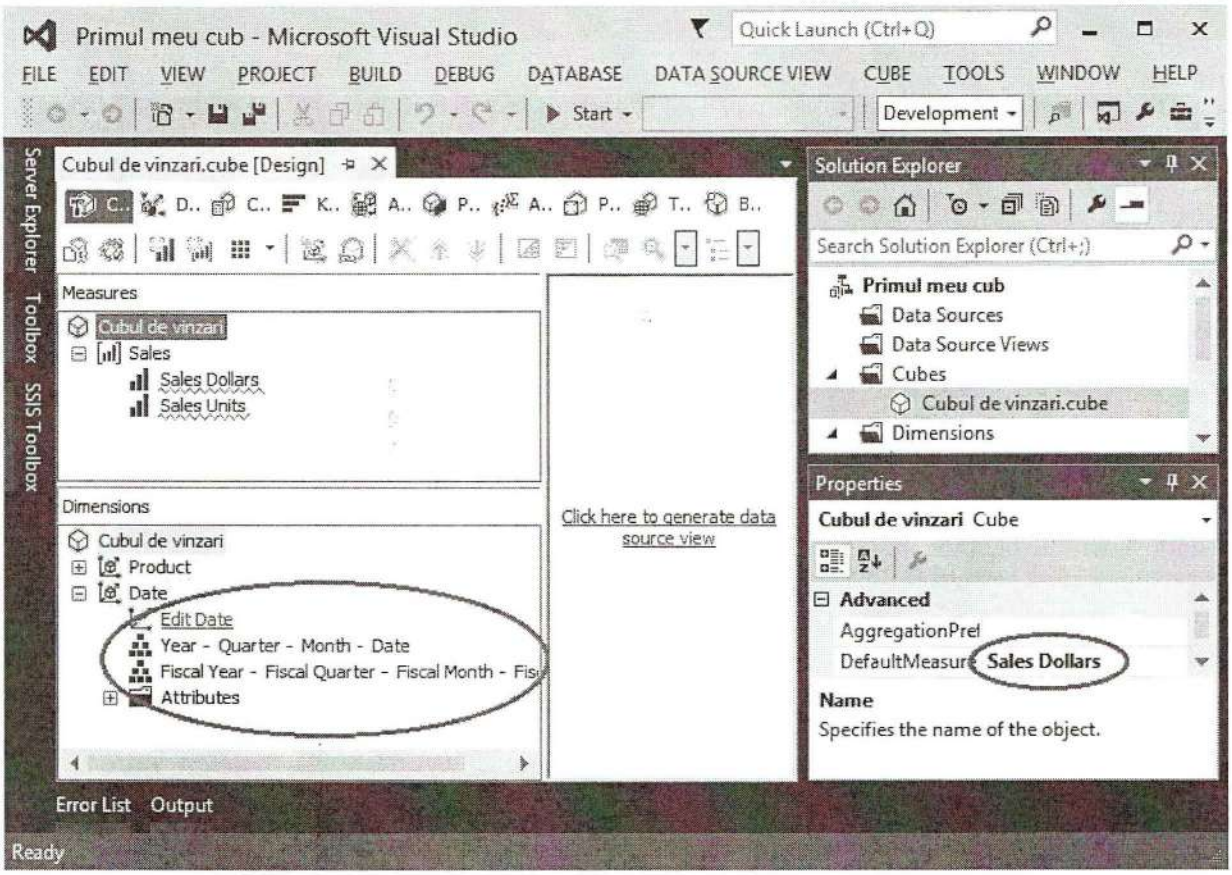
   Deoarece grupurile de măsuri implică o serie de proprietăți ale cubului, ultimele pot fi modificate numai la utilizarea formei de vizualizare ***Tree View.*** Proprietatea ***DefaultMeasure*** este asociată cu cubul și poate fi necesară modificarea ei, atunci când cubul a fost construit în baza unui șablon și se dorește excluderea măsurii implicite a șablonului din cub. În acest caz, măsura ***Reseller Sales Amount*** nu este una validă pentru cubul creat și, prin urmare, e necesară modificarea proprietății.

5. În fereastra ***Properties,*** se găsește proprietatea ***DefaultMeasure*** (în partea de sus a listei), și se selectează ***Sales Dollars*** din lista derulantă a proprietății.

   Trebuie menționat că în cazul în care această proprietate nu este resetată, cubul nu va putea fi procesat cu succes.

6. Se expandează dimensiunea ***Date*** din panoul ***Dimensions.***

   Ecranul va arăta, precum urmează:



În panoul ***Dimensions*** sunt prezente cele două ierarhii construite de asistentul ***Cube Wizard***:

- Year – Quarter – Month – Date
- Fiscal Year – Fiscal Quarter – Fiscal Month – Fiscal Day

Linkul ***Edit Date*** din capul listei de ierarhii permite deschiderea ***Dimension Structure*** pentru dimensiunea ***Date***. ***Dimension Structure*** este un spațiu de lucru în componența ***Dimension Designer,*** în care pot fi redactate proprietățile atributelor și ierarhiile într-o dimensiune dată.

7. Se expandează dimensiunea ***Product*** din panelul ***Dimensions*** și se extinde dosarul ***Attributes***.

În consecință, în panelul ***Dimensions*** sunt afișate toate atributele șablonului ***Adventure Works Standard Edition*** folosit pentru crearea cubului. Dacă nu se dorește includerea unora din aceste atribute în dimensiunea curentă, ele pot fi suprimate pe loc. Atributele unei dimensiuni pot fi modificate sau suprimate și în ***Dimension Designer.***

13.5. Crearea tabelelor-dimensiuni și tabelelor-fapte

La folosirea asistentului ***Cube Wizard*** pentru construirea unui cub fără sursa de date, este necesară crearea unei baze de date relaționale care va conține date ce vor fi încărcate în cub. Tabelele-dimensiuni și tabelele-fapte pot fi create în mod manual, dar o metodă mai simplă este folosirea asistentului ***Schema Generation Wizard***. Îndată ce schema este construită, datele pot fi inserate în tabelele-dimensiuni și tabelele-fapte cu scopul de a le putea vizualiza pe acestea în cub.

13.5.1. Generarea unei Scheme folosind asistentul Schema Generation Wizard

Asistentul ***Schema Generation Wizard*** ghidează procesul de construire a schemei relaționale. Dar, mai întâi, trebuie creată baza de date, în care asistentul va adăuga tabelele-dimensiuni și tabelele-fapte ale căror date vor fi încărcate în cub. Fiind construită structura relațională, asistentul adaugă un DSV bazat pe sursa de date specificată la proiectul realizat în ***Analysis Services***. Sursa de date definește informația de conexiune pentru sursa bazei de date, în timp ce DSV este reprezentarea structurilor și legăturilor dintre tabelele ce vor fi utilizate la modificarea cubului și la încărcarea datelor în cub.

Cu alte cuvinte, la această etapă se construiește baza de date relațională de procesare analitică on-line - ROLAP. Ea va fi folosită de cub, de baza de date OLAP.

În procedura următoare, se lansează asistentul ***Schema Generation Wizard***.

1. Se lansează ***Microsoft SQL Server Management Studio***. Se conectează la motorul de baze de date.
2. În fereastra ***Object Explorer,*** se aplică un click-drept pe dosarul ***Databases*** și apoi un click pe opțiunea ***New Database.***
3. Se inserează numele bazei de date, ***bd_simpla***, apoi se aplică un click pe butonul **OK**.
4. Se trece la ***Visual Studio*** (la proiectul nostru) și apoi în panelul ***Diagram*** de pe fila ***Cube Structure*** se face un click pe linkul ***Click Here To Generate Data Source View***.
5. În consecință, se deschide prima pagină a asistentului ***Schema Generation Wizard*** care se numește ***Welcome to the Data Source Wizard***. Aici se aplică un click pe butonul **Next**.

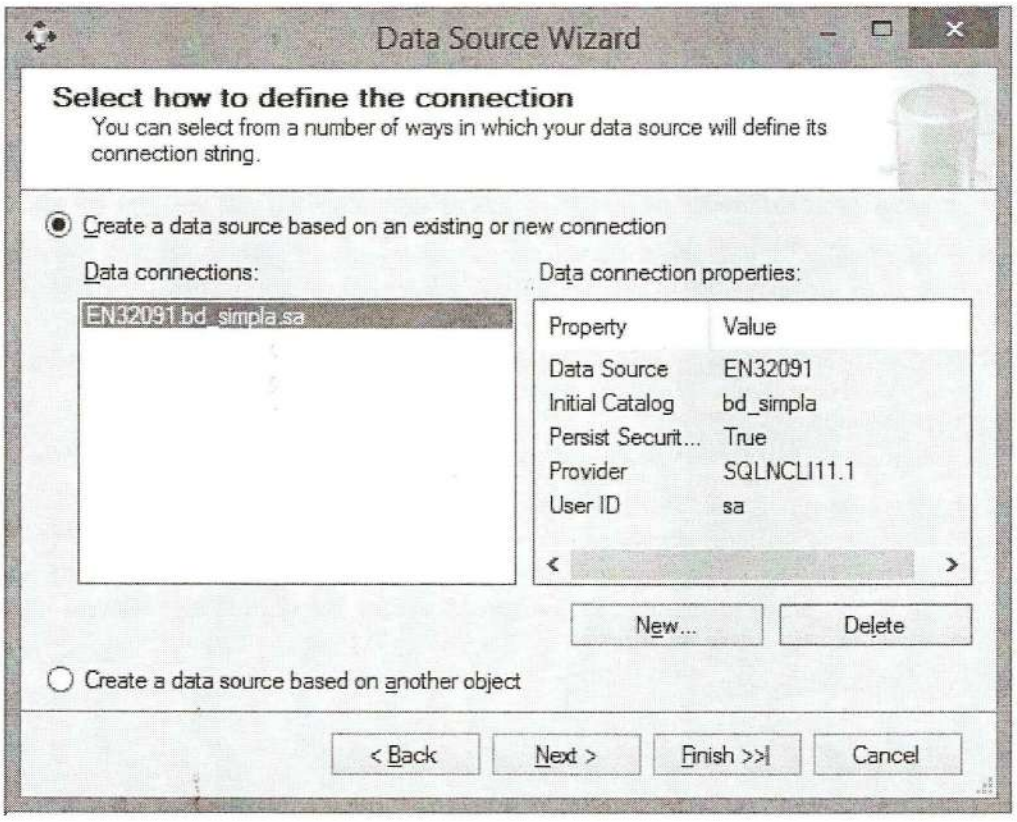
6. Apare pagina ***Specify Target*** a asistentului. În această pagină se specifică viziunea sursei de date în care asistentul va forma schema bazei de date a domeniului de interes. Deşi există posibilitatea de selectare a unei viziuni a sursei de date existente, de obicei, se recomandă crearea unei viziuni noi bazate pe o sursă de date. Poate fi creată sursa de date în baza unei conexiuni existente sau a unei conexiuni noi, sau în baza unui alt obiect.

   Asistentul ***Schema Generation*** generează schema pentru baza de date din domeniul de interes într-o bază de date, la care se referă sursa de date, precum şi a viziunii sursei de date. ***Schema Generation Wizard*** nu creează însăşi baza de date a domeniului; în schimb, asistentul creează schema relaţională pentru a susţine cuburile şi dimensiunile într-o bază de date existentă specificată.

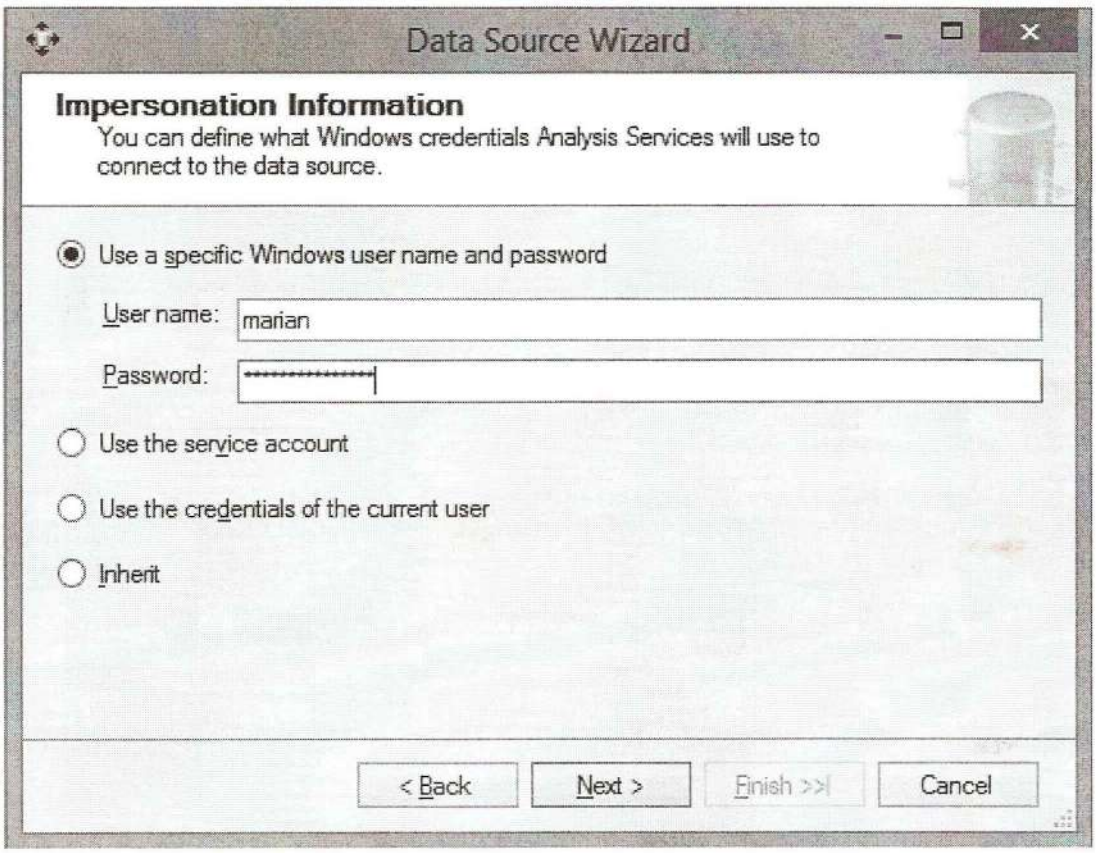
   Dacă sursa de date a fost adăugată anterior la proiect, ea poate fi selectată din câmpul derulant ***Data Source***. În cazul nostru, pentru a adăuga sursa de date, lângă câmpul de editare ***Data Source*** se apasă butonul **New.**

   În rezultat, se deschide pagina de bun venit (***Welcome to the Data Source Wizard***) a asistentului ***Data Source Wizard.*** Aici se aplică un click pe butonul **Next.**

6. În consecinţă, apare pagina ***Select how to define the connection*** a asistentului. În această pagină trebuie să fie bifată căsuţa ***Create a data source based on an existing or new connection*** şi se apasă butonul **New.**

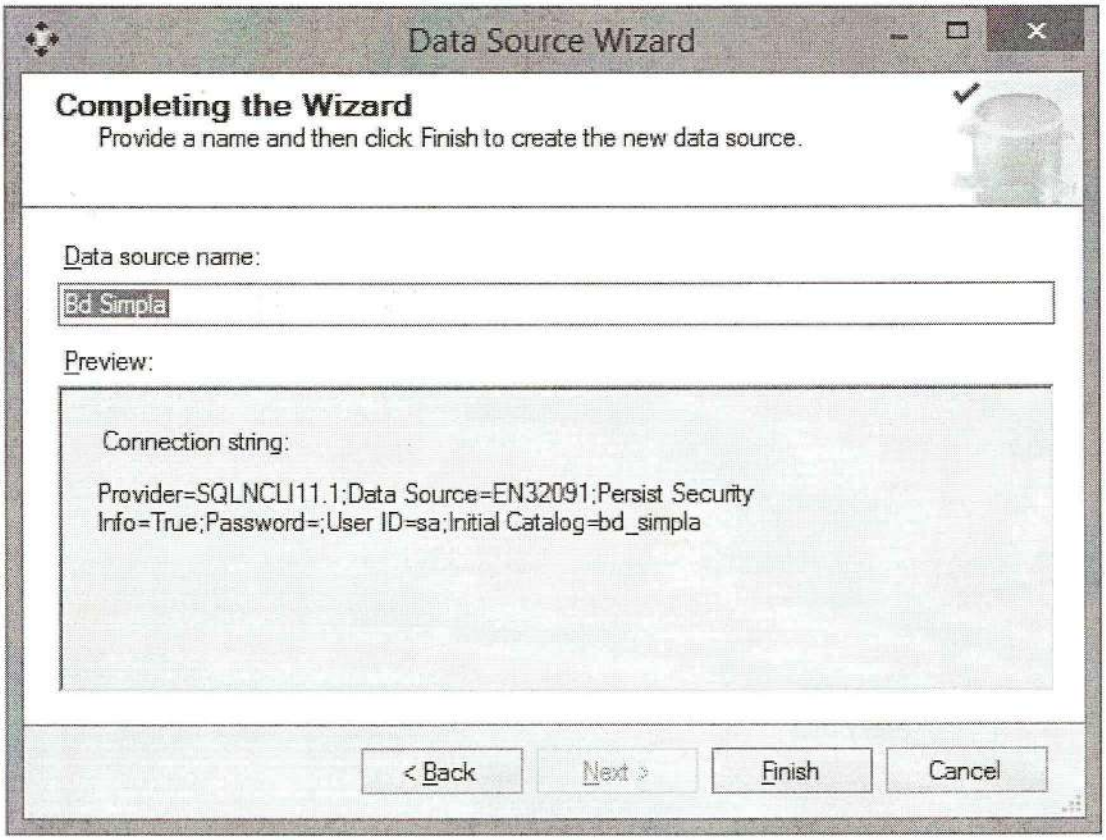


7. Apare caseta de dialog ***Connection Manager*** cunoscută, deja, din capitolele precedente.
8. Se selectează baza de date ***bd_simpla*** din meniul derulant ***Select Or Enter A Database Name***, se testează conexiunea şi se apasă butonul **OK**. Apoi în pagina deschisă a asistentului, se apasă **Next**.
9. Pe pagina ***Impersonation Information*** se definesc parametrii de autentificare ce vor fi utilizaţi pentru conectarea ***Analysis Services*** la sursa de date.

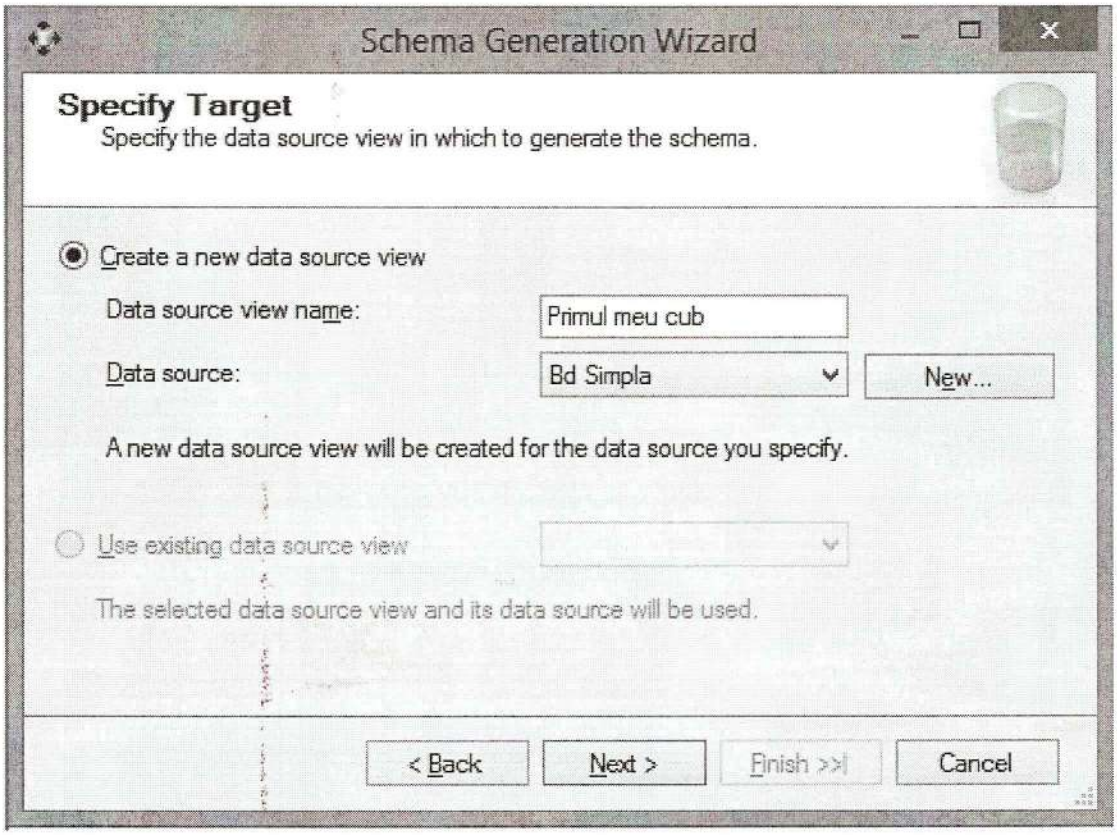


Din patru variante posibile de autentificare, în cazul nostru, se bifează ***Use a specific Windows user name and password.*** În câmpurile respective se specifică numele utilizatoruli şi parola pentru un login care va fi capabil să acceseze baza de date ***bd_simpla*** din ***Analysis Services***. Apoi se aplică un click pe **Next**.

Următoarea pagină - ***Completing the Wizard,*** se utilizează pentru vizionarea proprietăţilor de conexiune la sursa de date şi specificarea numelui noii surse de date. Valoarea implicită a câmpului ***Data Source Name*** este numele bazei de date selectate, ***bd_simpla.*** Proprietăţile conexiunii sunt afişate pe panelul ***Preview***.



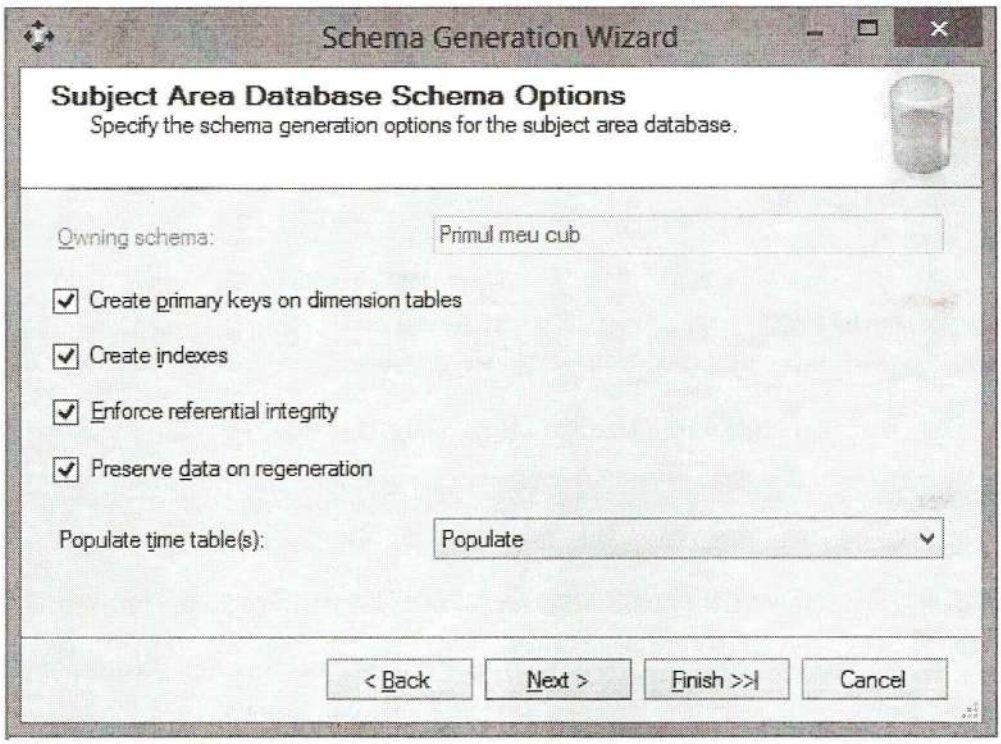
10. Pentru a termina lucrul cu asistentul ***Data Source Wizard*** se apasă butonul **Finish.** Ecranul va arăta, precum urmează:



Acum, când sursa de date este accesibilă, poate fi continuat procesul de generare a schemei cu asistentul ***Schema Generation Wizard***. Sistemul revine la pagina ***Specify Target***.

11. Se apasă butonul **Next**.

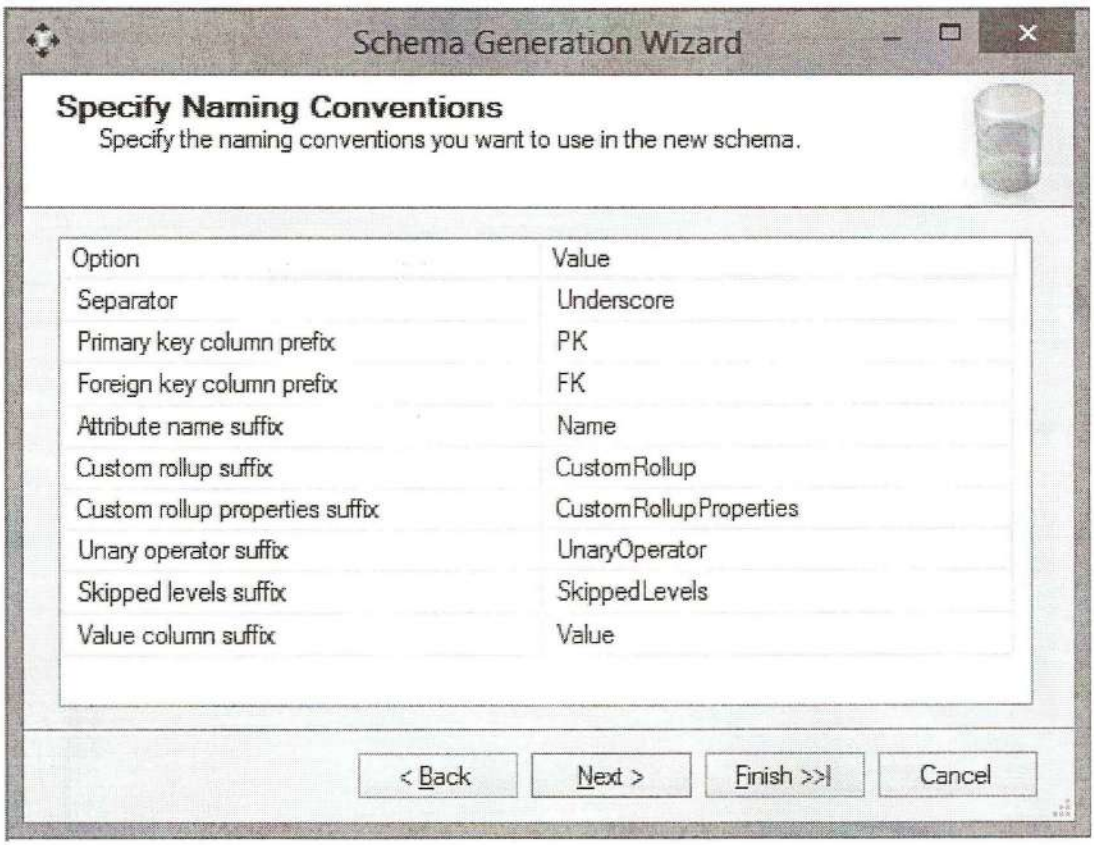
Următoarea pagină ***Subject Area Database Schema Options*** se utilizează pentru controlul modului de generare a schemei, precum şi pentru specificarea modului de păstrare a datelor.



- Câmpul ***Owning schema*** specifică numele schemei în cadrul domeniului nou în baza de date.
- Dacă este bifată opțiunea ***Create primary keys on dimension tables***, sistemul creează cheile primare pe tabele-dimensiuni în schema generată. Nici un index nu este generat, dacă nu este selectată această opțiune.
- ***Create indexes*** creează, în schema generată, indecşi pe coloanele cheii externe.
- ***Enforce referential integrity*** forțează integritatea referențială în cadrul schemei generate. Dacă nu este selectată această opțiune, legăturile sunt create, dar nu sunt constrânse cu acest tip de integritate.
- ***Preserve data on regeneration*** păstrează datele în baza de date a domeniului de interes, după ce asistentul şi-a terminat activitatea. Dacă această opțiune nu este selectată, toate datele din baza de date a domeniului pot fi şterse fără avertisment.
- Câmpul de editare ***Populate time table(s)*** specifică modul în care asistentul populează tabele temporale. Meniul derulant conține valorile posibile pentru această opțiune.

12. În cazul nostru, se consideră implicit că toate opțiunile sunt selectate, deoarece acestea sunt recomandate în calitate de strategii pentru proiectarea schemei relaționale.

Se apasă butonul **Next**. Asistentul ***Schema Generation Wizard*** ia următoarea formă:

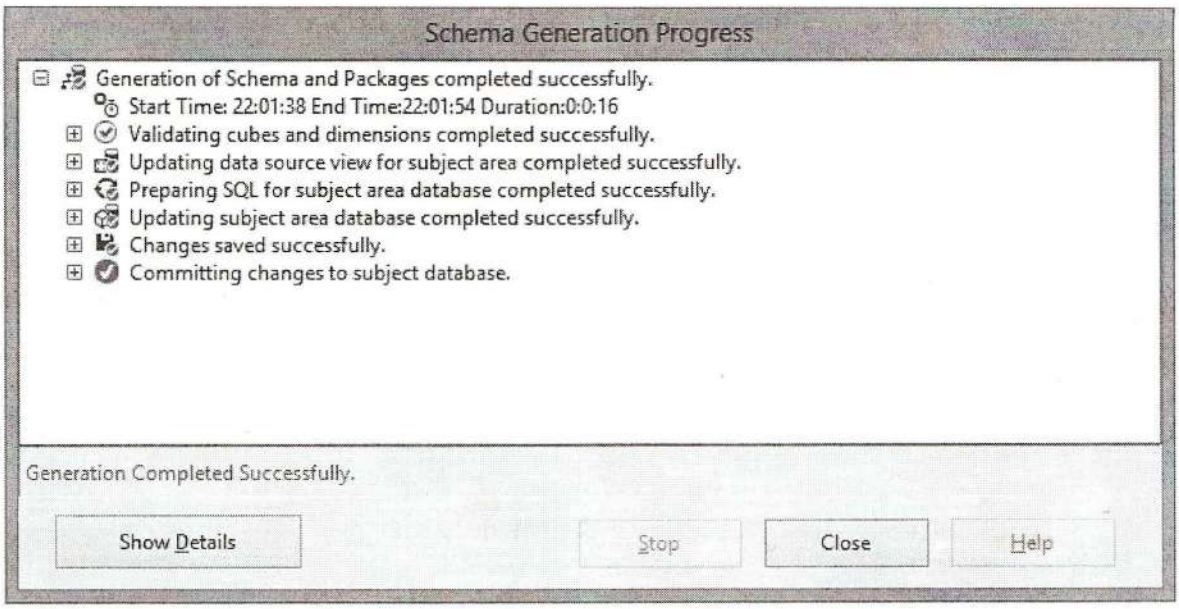


Pe pagina ***Specify Naming Conventions***, pot fi ajustate convențiile de denumire folosite de asistent, dacă sunt standarde care se doresc utilizate.

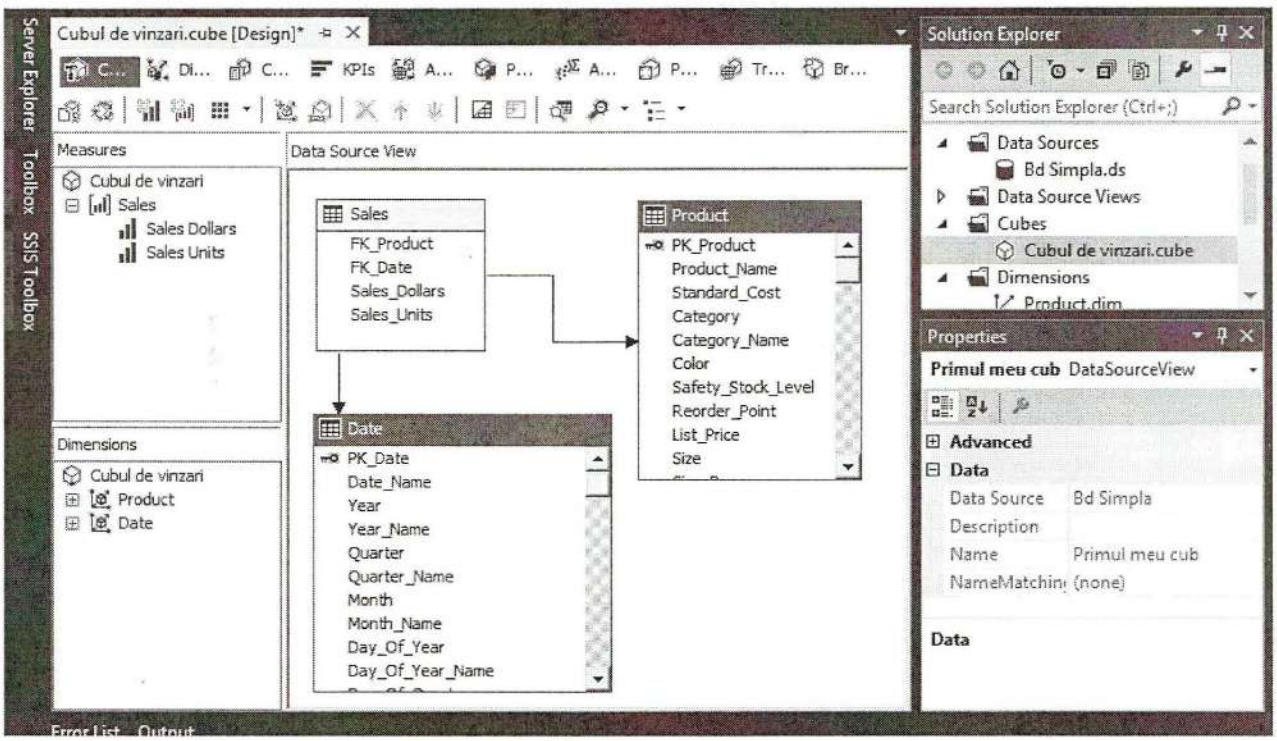
- Delimitatorul implicit este sublinierea, dar poate fi folosit un spațiu sau chiar niciun delimitator. În calitate de prefixe pentru cheile primare şi externe, se folosesc valorile implicite PK şi FK, respectiv, care, de asemenea, pot fi modificate.
- În cazul în care un atribut are atât o coloană-cheie, cât şi o coloană-nume, denumirea implicită a sufixului numelui de coloană este ***Name***, care poate fi schimbat, dacă nu există alt sufix cu aceeaşi valoare.
- Restul convențiilor asupra numelor sunt aplicate la sufixele utilizate pentru coloanele ce suportă caracteristicile ***Analysis Services*** speciale: ***custom rollup, custom rollup properties, unary operators*** şi ***skipped levels***. Pot fi schimbate valorile implicite ale sufixelor pentru toate aceste tipuri de coloane, atunci când fiecare valoare este unică printre toate sufixele.

13. Se face un click pe butonul **Next**, şi apoi pe **Finish**.

Fereastra ***Schema Generation Progress*** reprezintă statutul de procesare şi oferă detalii asupra paşilor parcurşi. Paşii parcurşi sunt grupați pe categorii într-un arbore. Procesul poate fi oprit apăsând butonul **Stop**. La terminarea prelucrării, butonul **Close** devine activ.



14. Se face un click pe butonul **Close**. Ecranul ia forma următoare:



13.5.2. Încărcarea datelor în schema relațională

Acum când tabelele-dimensiuni şi tabelele-fapte sunt în baza de date creată – ***bd_simpla***, este posibilă popularea lor cu date cu scopul de procesare a cubului. Procesarea cubului este o operație de încărcare a datelor în cub şi în fişierele-dimensiuni din tabelele relaționale. Metodologia de pregătire a datelor pentru încărcarea în magazie iese din limitele acestei lucrări. Pentru a duce la bun sfârşit procedura de încărcare, e necesară copierea unei submulțimi de date din baza de date ***AdventureWorksDW2014*** în baza de date nou-creată, ***bd_simpla.***

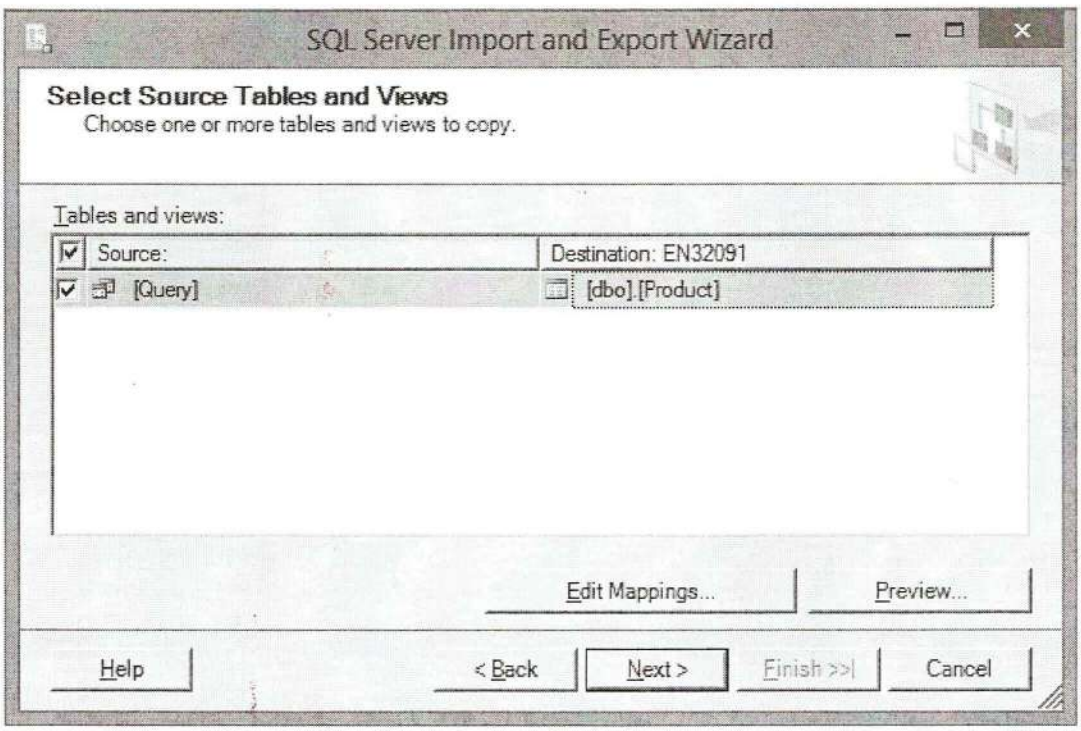
Următoarea procedură, va efectua copierea datelor din tabelele ***DimProduct*** şi ***FactResellerSales*** din baza de date ***AdventureWorksDW2014*** în baza de date – ***bd_simpla,*** folosind asistentul ***SQL Server Import and Export Wizard***. Vor fi populate doar unele coloane din baza de date ***bd_simpla,*** suficiente pentru a demonstra utilitatea tehnologiei OLAP.

1. Se deschide SQL Server Management Studio.
2. Se accesează motorul bazei de date
3. În ***Object Explorer*** se aplică un click-drept pe baza de date ***AdventureWorksDW2014,*** se selectează ***Tasks***, apoi ***Export Data***.
4. Apare pagina de bun venit a asistentului ***SQL Server Import and Export Wizard*** . Se acționează butonul **Next**.
5. În pagina ***Choose a Data Source,*** pentru conectarea la ***AdventureWorksDW2014,*** câmpurile ***Data Source***, ***Server name*** și ***Database*** se completează în mod implicit. Se acționează butonul **Next**.
6. Din lista derulantă a câmpului ***Database*** de pe pagina ***Choose a Destination*** se alege baza de date ***bd_simpla***. Se acționează butonul **Next**.
7. Se bifează radio-butonul Write A Query To Specify The Data To Transfer de pe pagina Specify Table Copy Or Query. Se acționează butonul Next.
8. Se inserează următoarea interogare în câmpul de instrucțiuni SQL din pagina ***Provide a Source Query***:

```
SELECT ProductKey, EnglishProductName, [Weight] FROM DimProduct
```

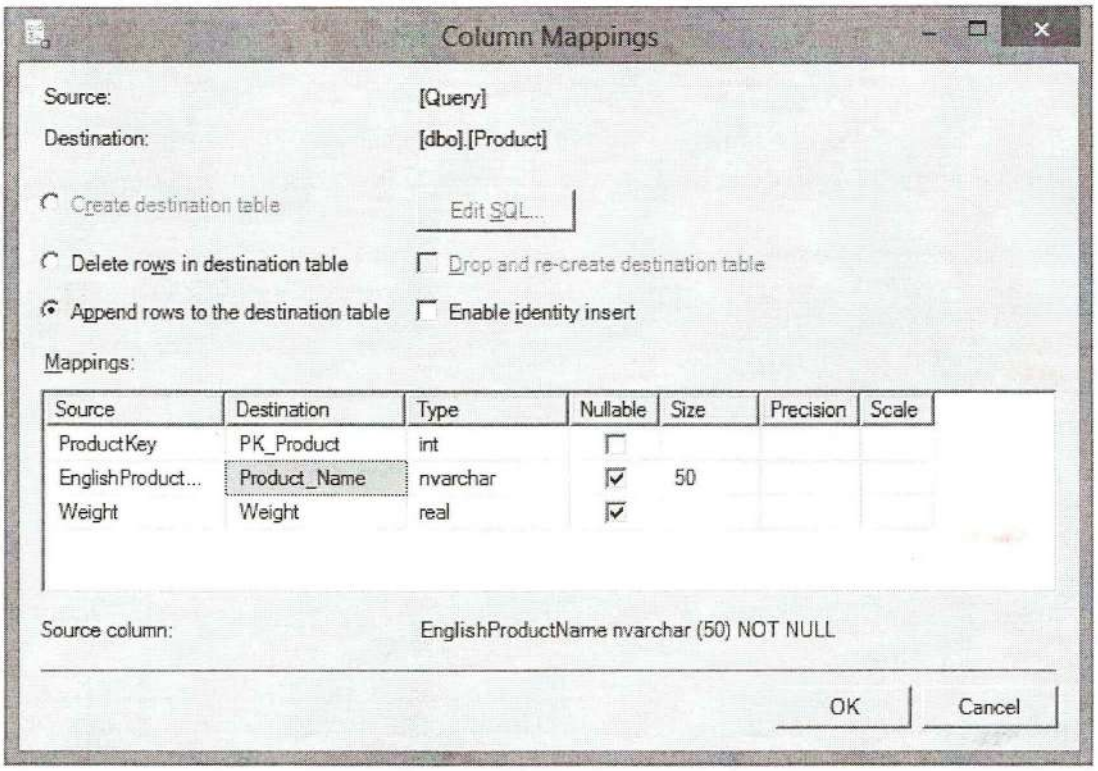
Se acționează butonul **Next**.

9. Se selectează opțiunea ***[dbo].[Product]*** din lista derulantă ***Destination*** din pagina ***Select Source Tables and Views***.



10. Din partea inferioară a paginii se acționează butonul **Edit Mappings**. În cutia de dialog apărută ***Column Mappings,*** pentru fiecare element din coloana ***Source,*** se selectează din lista derulantă a coloanei ***Destination*** opțiunea respectivă, precum e prezentat în următorul tabel:

Source	Destination
ProductKey	PK_Product
EnglishProductName	Product_Name
Weight	Weight



11. Pentru a închide cutia de dialog ***Column Mappings*** se acționează butonul **OK**. Apoi se apasă butonul **Next**.

    Pe pagina apărută, ***Review Data Type Mapping,*** se lasă totul fără schimbări și se apasă **Finish**.

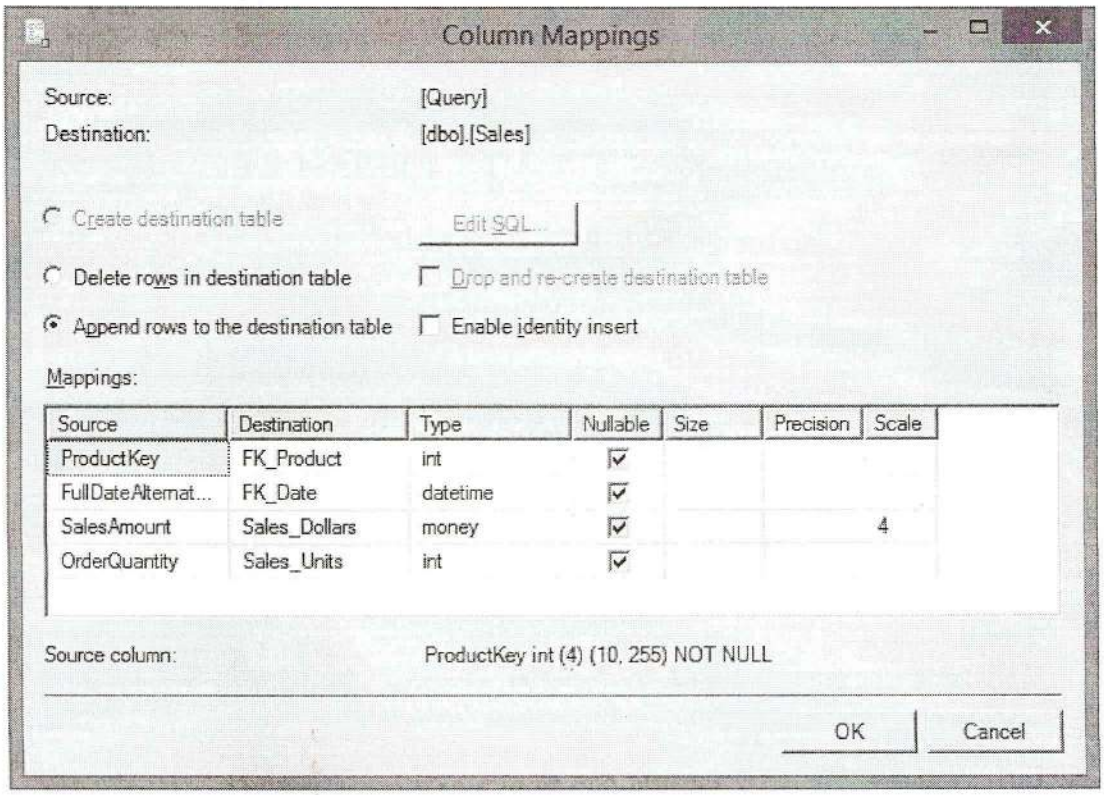
    Apoi se apasă butonul **Finish** în fereastra ***Complete the Wizard***. Statutul operațiilor de copiere vor fi prezentate pe pagina ***Performing Operation***.
12. După efectuarea operațiilor, se acționează butonul **Close**.
13. Se aplică un click-drept pe baza de date ***AdventureWorksDW2014,*** se selectează ***Tasks***, apoi ***Export Data***.
14. Se acționează butonul **Next** de două ori. Se alege baza de date ***bd_simpla*** din lista derulantă, de pe pagina ***Choose a Destination***. Se acționează butonul **Next**.
15. Se acționează radio-butonul *Write A Query To Specify The Data ToTransfer* de pe pagina *Specify Table Copy Or Query*. Se acționează butonul Next.
16. În pagina ***Provide a Source Query,*** se inserează următoarea interogare în câmpul de instrucțiuni SQL:

```
SELECT ProductKey, FullDateAlternateKey,
       SalesAmount, OrderQuantity
FROM   FactResellerSales, DimDate
WHERE  DimDate.DateKey = FactResellerSales.OrderDateKey
```

    Se acționează butonul **Next**.
17. În pagina ***Select Source Tables and Views*** din lista derulantă ***Destination*** se selectează opțiunea ***[dbo].[Sales]***.
18. Din partea inferioară a paginii se acționează butonul **Edit Mappings**. În cutia de dialog ***Column Mappings,*** pentru fiecare element din coloana ***Source,*** se selectează opțiunea respectivă din meniul derulant al coloanei ***Destination,*** precum este prezentat în următorul tabel:

Source	Destination
ProductKey	FK_Product
FullDateAlternateKey	FK_Date
SalesAmount	Sales_Dollars
OrderQuantity	Sales_Units

Ecranul va arăta, precum urmează:



19. Pentru a închide cutia de dialog ***Column Mappings*** se acționează butonul **OK**. Apoi se apasă **Next**, apoi butonul **Finish** de două ori. La sfârșit se acționează butonul **Close**.

    Acum, datele s-au încărcat în ***bd_simpla.*** Tabelul-dimensiuni, ***dbo.Date***, a fost populat, în mod automat, cu ajutorul asistentului ***Schema Generation Wizard***. Apoi, s-au efectuat două operații de export al datelor din baza de date ***AdventureWorksDW2014*** în tabelele ***dbo.Product*** și ***dbo.Sales***.

    Însă, datele încă n-au fost încărcate în cub. Acest scop va fi atins în procedura următoare.

13.5.3. Procesarea dimensiunilor vs procesarea cuburilor

Procesarea unei dimensiuni și procesarea unui cub sunt două sarcini distincte.

În cazul procesării unei dimensiuni, ***Analysis Services*** citește informația din tabelele-dimensiuni și construiește o hartă a dimensiunilor bazată pe fiecare atribut. În limitele hărții, fiecare valoare din coloanele tabelului-dimensiuni este un atribut-membru stocat într-o ierarhie cu două nivele pentru dimensiunea curentă. De exemplu, fie că ***Washington*** și ***Oregon*** sunt membri ai atributului ***State*** în dimensiunea ***Geography***. Harta păstrează acești membri în ierarhia ***State*** de atribute, în felul următor: ***All_Washington*** și ***All_Oregon***. Păstrând fiecare membru al ierarhiei atributului pe două nivele, ***Analysis Services*** poate utiliza harta pentru a găsi legăturile dintre membri în mod rapid, atunci când se interoghează cubul. Evident că harta dimensiunilor este un obiect comprimat și foarte eficient.

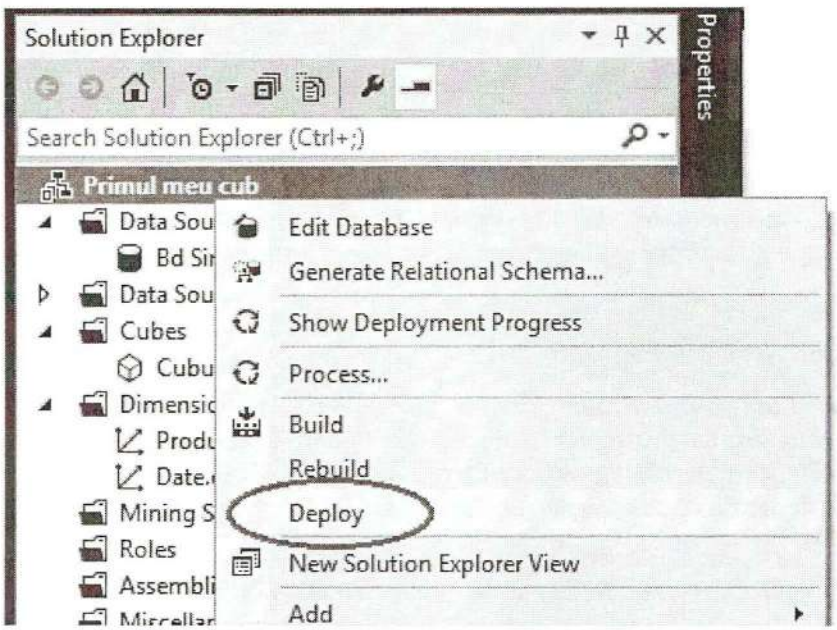
În cazul procesării unui cub, ***Analysis Services,*** mai întâi, îmbină hărțile dimensiunilor din toate dimensiunile utilizate în cub într-o hartă multidimensională a cubului. Apoi, se citesc înregistrările din

tabelele-fapte stocând valorile acestora în aria de stocare a datelor. Aria de stocare este eficient organizată şi nu se oferă spaţiu pentru acele combinaţii de chei, ce nu conţin nicio valoare.

13.5.4. Desfăşurarea şi procesarea Cubului

În această procedură, componenta ***Analysis Server*** va desfăşura proiectul ***Primul meu cub*** pe care-l va procesa şi-l va face accesibil pentru vizualizare.

1. În componenta ***Visual Studio,*** se aplică un click-drept pe proiectul ***Primul meu cub*** din fereastra ***Solution Explorer***, apoi în meniul de context apărut se acţionează opţiunea ***Deploy***.



În fereastra output apare informaţia despre starea curentă a procesului de desfăşurăre a cubului şi rezultatele executării etapelor principale de desfăşurare.

Fereastra ***Deployment Progress,*** care se află în partea dreaptă a mediului de proiectare, monitorizează executarea activităţilor curente a componentelor procesului de desfăşurare şi construieşte un arbore de obiecte desfăşurate.

Informaţia detaliată despre desfăşurarea fiecărei componente a cubului poate fi obţinută expandând nodul respectiv al arborelui din această fereastra.

În cazul executării normale a procesului de desfăşurare, în final, în partea de jos a ferestrei ***Deployment Progress*** va apărea mesajul: ***Deployment Completed Successfully***.

2. Implicit, desfăşurarea din cadrul ***Visual Studio*** procesează atât dimensiuni, cât şi cuburi în cazul în care acestea sunt trecute şi examinate, pentru prima dată, de ***Analysis Server***. La procesarea unui cub, ***Analysis Server*** copie valorile detaliate din tabelele-fapte într-o structură de date proprietar.

   După executarea cu succes a procesului de desfăşurare, în meniul ***File,*** se aplică un click pe opţiunea ***Close Project***.

Astfel, folosind asistentul ***Cube Wizard*** şi şablonul oferit de ***Analysis Services***, există posibilitatea de proiectare a unui cub independent de sursa existentă de date. Cuburile şi dimensiunile create, utilizând asistentul, nu întotdeauna pot reprezenta exact ceea de ce a fost nevoie. Structura cubului şi structurile dimensiunilor pot fi modificate, suprimând obiectele şi schimbând proprietăţile. Fiind făcute aceste modificări, deja, pot fi utilizaţi asistenţii ***Schema Generation Wizard*** şi ***SQL Server Import and Export Wizard*** pentru a genera tabelele şi a încărca datele în acestea cu scopul de a procesa cubul.

13.6. Interogarea cubului, folosind limbajul Multidimensional Expressions

13.6.1. Limbajul Multidimensional Expressions

Multidimensional Expressions (MDX) este un limbaj complet, bazat pe instrucțiuni, folosit pentru definirea, lucrul și căutarea datelor în obiectele multidimensionale în cadrul sistemului ***Microsoft SQL Server 2014 Analysis Services (SSAS).***

Multidimensional Expressions este un standard al limbajelor de interogare orientat spre analiză. Sintaxa acestui limbaj este similară, în multe privințe, cu sintaxa SQL. Cu toate acestea, MDX nu este o extindere a limbajului SQL. În realitate, unele funcționalități oferite de MDX sunt susținute și de SQL, dar nu atât de eficient și intuitiv.

Scopul interogării MDX constă în extragerea valorilor din cubul OLAP și plasarea lor într-un raport. În timp ce cubul are dimensiuni, raportul nu le posedă. Acesta e constituit din axe (de obicei, axa-rânduri, axa-coloane și axa-filtru). Axa poate conține etichete dintr-una sau mai multe dimensiuni. Cubul conține toate valorile posibile din toți membrii tuturor ierarhiilor de atribute ale tuturor dimensiunilor. Un raport conține doar valorile selectate din nivelele selectate ale ierarhiilor de atribute selectate ale dimensiunilor selectate. Prin urmare, interogarea MDX este o instrucțiune pentru extragerea raportului din cub.

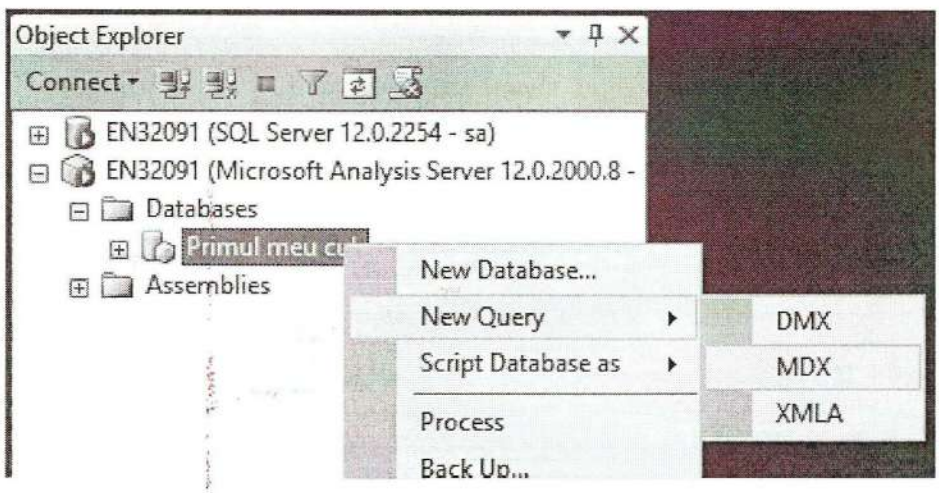
13.6.2. Executarea interogărilor MDX

SQL Server Management Studio oferă o interfață pentru scrierea interogărilor MDX, rularea acestora și examinarea rezultatelor reprezentate în formă de grid. Fereastra interogării MDX include un panel pentru metadate, care permit vizualizarea ierarhiilor și membrilor unui cub, inserarea în expresiile MDX a numelor de dimensiuni, de atribute și de membri.

Prin analogie cu interogările SQL, e ușor de recunoscut clauzele folosite în interogările MDX: *SELECT*, *FROM* și *WHERE*. Clauza *SELECT* definește ce membri vor fi incluși și unde vor fi plasați – în rânduri sau coloane. Clauza *FROM* denumește cubul, iar clauza opțională *WHERE* restricționează valorile returnate de interogare pentru membrii enumerați.

Cu scopul de a face cunoștință cu interfața de interogări a sistemului ***SQL Server Management Studio*** și cu sintaxa MDX, în continuare, sunt executate câteva interogări MDX simple.

1. Se lansează sistemul ***SQL Server Management Studio***. Se conectează serviciul ***Analysis Services.***
2. În ***Analysis Server,*** se expandează dosarul ***Databases,*** apoi pe baza de date necesară se face un click-drept, după care se alege opțiunea ***New Query***, apoi ***MDX***



În fereastra de interogări MDX, pot fi explorate metadatele pentru cubul ***Cubul de vinzari*** de pe fila ***Metadata***, afișată în panelul din stânga. Interogările MDX sunt afișate în câmpul de interogare din partea dreaptă. Pot fi executate mai multe interogări MDX împreună, după cum sunt executate mai multe interogări SQL, adică indicând comanda GO între acestea. Panelul de

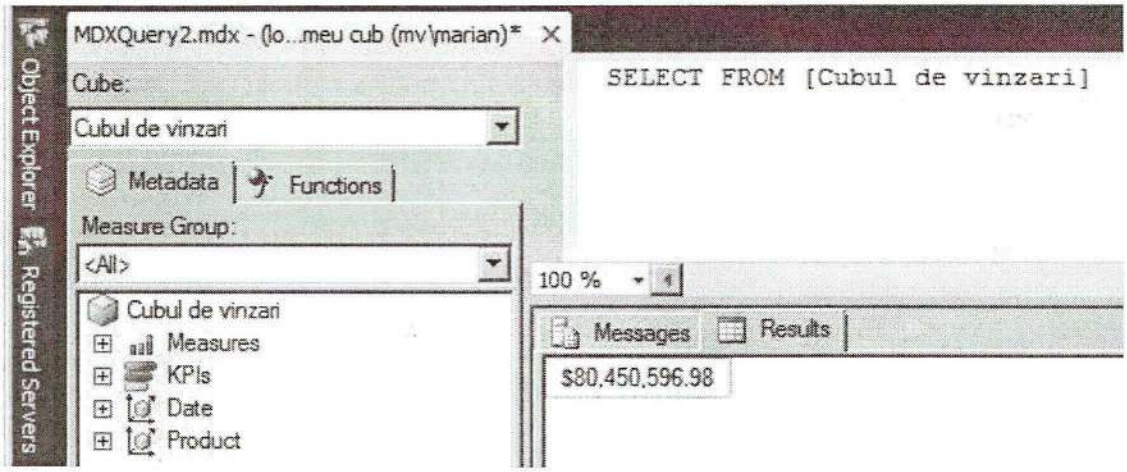
jos va apărea în urma executării unei interogări MDX, în care se va afişa rezultatul obţinut.

3. Ca rezultat al primei interogări, prezentate în figura de mai jos, în câmpul de rezultate apare un singur număr - $80,450,596.98 . Acest număr reprezintă totalul de vânzări ale mărfii (***[Sales Dollars]***).

   Această interogare este cea mai simplă interogare ce poate fi executată. Deoarece nu este specificată nici măsura implicită a cubului, MDX selectează măsura implicită definită în cub, care poate fi vizualizată în fila ***Cube Structure*** din ***Visual Studio***.

   Dacă măsura implicită nu este definită explicit, atunci, prima măsură creată în cub va fi considerată de sistem ca implicită. În plus, deoarece interogarea nu specifică membrii unei dimensiuni, aceasta foloseşte membrul implicit pentru fiecare ierarhie de atribute din fiecare dimensiune, care, la moment, este membrul ***All*** şi extrage o singură valoare din cub reprezentată de tuplul din fereastra-rezultat.

   Un membru implicit există pentru orice ierarhie de atribute a cubului, indiferent dacă este explicit aceasta definită sau nu. În calitate de membru implicit este membrul ***All*** al unei ierarhii de atribut, dar se poate seta un alt membru prin atribuirea unei valori proprietăţii ***DefaultMember*** a atributului.



4. În a doua interogare, mulţimea de pe axa-coloane constă din două unităţi. Fiecare unitate corespunde unui tuplu şi fiecare tuplu, la rândul său, conţine un singur membru.

```
SELECT
{[Measures].[Sales Units],[Measures].[Sales Dollars]}ON COLUMNS
   FROM [Cubul de vinzari]
```

100 %

Messages | Results

Sales Units	Sales Dollars
214,378	$80,450,596.98

Pot fi adăugaţi alţi membri din ***Measures***, dar mulţimea trebuie luată în acolade - *{[mulţimea]}*. Acoladele nu sunt necesare în cazul utilizării funcţiilor asupra mulţimilor, dar ele devin necesare la crearea unei mulţimi constituite din doi sau mai mulţi membri.

Adresarea la o componentă din măsură se execută cu ajutorul enumerării prin punctul lui în ierarhie. Iată unele exemple:

`[Date]` – măsură ***Date***

`[Date].[Year-Quarter-Month-Date]` – ierarhie ***Year-Quarter-Month-Date***

[Date].[Year-Quarter-Month-Date].[Year] – nivel ***Year***

[Date].[Year-Quarter-Month-Date].[Year].&[2013-01-01] – membru dată.

Includerea fiecărui element în paranteze pătrate nu este obligatorie, dar deoarece mai multe obiecte într-un cub conțin în numele lor spații și simboluri speciale, utilizarea parantezelor pătrate este binevenită. Simbolul ampersend (&) înainte de numele membrului indică la referința după cheie. Fiecare membru al măsurii poate fi adresat atât în baza unui nume, cât și în baza cheii. Ultima variantă de adresare este recomandată, deoarece identificatorul cheii poate să se deosebească de valoarea reală a câmpului.

5. În ultima imagine, sunt reprezentate două interogări. Prima interogare afișează informația statistică despre suma de bani și de unități de marfă vândută, grupată pe ani. Clauza *NON EMPTY* indică sistemului să afișeze numai valorile nevide pentru dimensiunea ***[Year]***.

   În a doua interogare, în calitate de coloană, este specificat numai anul 2011 și, în calitate de filtru, este specificat un tip de produs anumit. Ca rezultat al executării acestei interogări, se obține statistica despre suma de bani și de unități de marfă specificată în condiția *WHERE*, care este vândută în anul 2011 (expresia ***[Product].[Product].&[310]*** este echivalentă cu ***'Road-150 Red, 62'***).

```
SELECT {[Measures].[Sales Units],[Measures].[Sales Dollars]} ON COLUMNS,
       NON EMPTY [Date].[Year - Quarter - Month - Date].[Year] ON ROWS
 FROM [Cubul de vinzari]
GO
SELECT {[Measures].[Sales Units],[Measures].[Sales Dollars]} ON COLUMNS,
       [Date].[Year - Quarter - Month - Date].[Year].&[2011-01-01T00:00:00] ON ROWS
 FROM [Cubul de vinzari] WHERE [Product].[Product].&[310]
```

100 %

Messages | Results

	Sales Units	Sales Dollars
Calendar 2010	820	$489,328.58
Calendar 2011	28,572	$18,192,802.71
Calendar 2012	81,328	$28,193,631.53
Calendar 2013	103,658	$33,574,834.16

	Sales Units	Sales Dollars
Calendar 2011	261	$560,357.08

După cum se vede din exemplele de mai sus, limbajul de interogări MDX reprezintă un instrument util pentru obținerea rapidă și simplă a informațiilor statistice din cuburile OLAP. Bineînțeles că interogările MDX pot avea un caracter complex, în funcție de scopurile și de structura cubului în cauză.

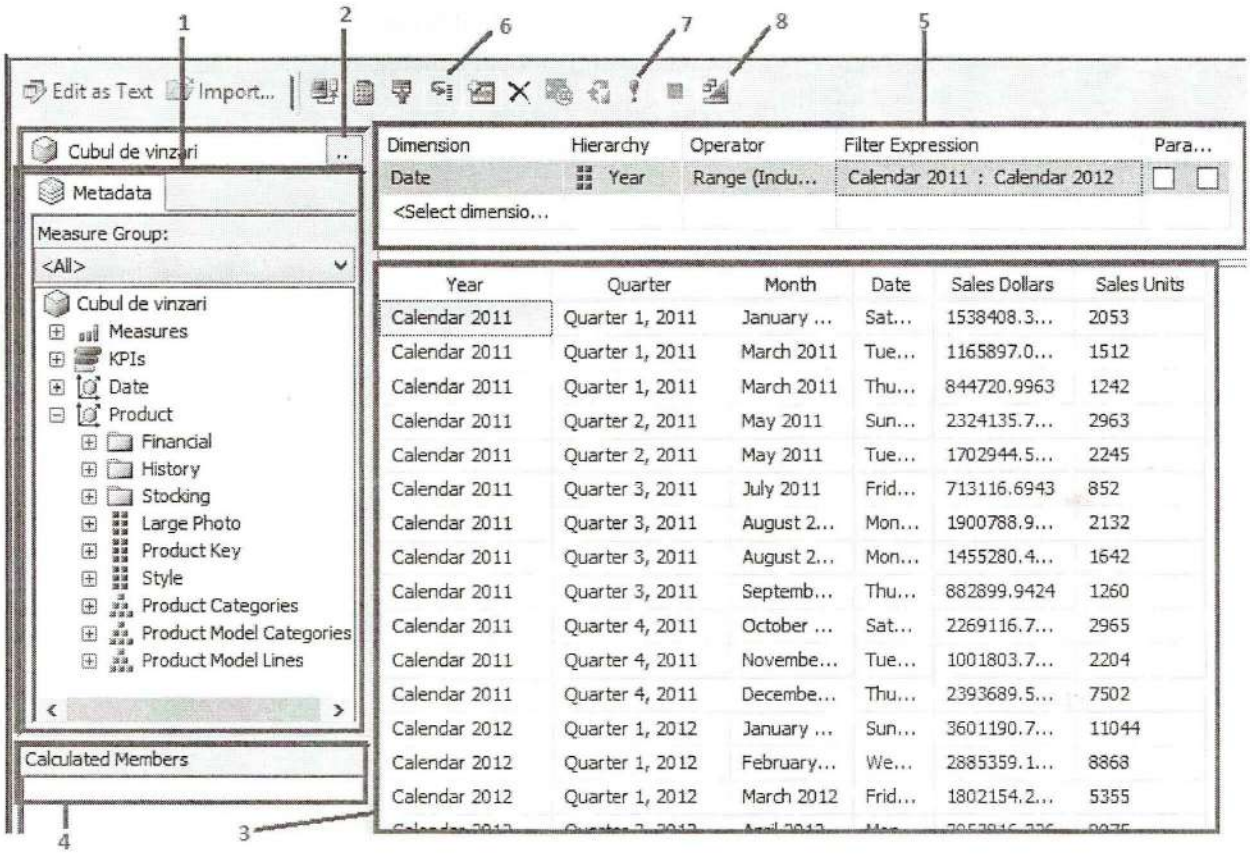
13.6.3. Explorarea dimensiunilor și măsurilor

Pentru explorarea dimensiunilor și măsurilor unui cub poate fi utilizată fila ***Browser*** din ***Cube Designer*** al mediului ***Visual Studio***. Același navigator este disponibil și în mediul ***Microsoft SQL Server Management Studio***: în ***Object Explorer*** se face un click-drept pe cubul necesar și din meniul apărut se alege ***Browse***.

În ***SQL Server 2014 Analysis Services***, navigatorul cubului a fost integrat cu constructorul de interogări MDX și oferă o interfață grafică pentru generarea interogărilor MDX, filtrarea și decuparea cuburilor, precum și aplicarea operațiilor *drill-down* în ierarhiile dimensionale.

Fila ***Browser*** are două regimuri de funcționare: regimul *proiectare* (*Design Mode*) și regimul *interogare* (*Query Mode*). În ambele regimuri, pot fi utilizate obiectele din panoul ***Metadata*** pentru explorarea cubului. Există posibilitatea de glisare a membrilor din panoul ***Metadata*** în zona de interogare, pentru a construi interogări MDX de preluare a datelor dezirabile.

Imaginea de mai jos prezintă interfața navigatorului în regimul de proiectare grafică:



Navigatorul cubului conține 5 paneluri principale:

1. Panelul ***Metadata***. În formă de arbore sunt afișate măsurile și dimensiunile cubului.
2. Panelul de ***selectare a cubului***. Apăsând butonul de pe acest panel, poate fi ales cubul necesar.
3. Panelul ***Data***. În acest panel sunt afișate datele returnate de interogare.
   - Se glisează obiectul ***Measures*** din arborele de metadate al cubului, poziționat în partea stângă a ecranului, spre panelul de date.
   - Valorile ***Sales Dollars*** și ***Sales Units*** din panelul de date reprezintă totalurile generale ale acestor două măsuri ale cubului. Aceste valori sunt echivalente cu utilizarea funcției de agregare ***SUM*** aplicată pentru fiecare măsură-coloană din tabelul-fapte.
   - Măsurile pot fi glisate una câte una în panelul de date, dacă obiectul ***Measures*** este expandat în arborele de metadate al cubului. Sau ele pot fi toate glisate în panelul de date prin glisarea obiectului ***Measures***.
   - În panelul ***Metadata*** al cubului se expandează dimensiunea ***Date***, apoi se glisează, în panelul de date, ierarhia de dimensiuni ***Year-Quarter-Month-Date***. Valorile agregate ale masurilor sunt repartizate, pentru fiecare înregistrare, valorilor dimensiunii.
4. Similar cu panelul ***Metadata***, panelul ***Calculated Menbers*** conține măsurile valorile cărora reprezintă rezultatul unor operații de calcul asupra măsurilor reale. Ca analogie pot servi câmpurile calculate din modelul relațional.
5. Panelul ***Filter*** funcționează similar cu expresia din clauza *WHERE* a limbajului MDX. În calitate de filtru pot fi selectate numai dimensiuni.

   În exemplul de mai sus, în calitate de filtru a fost aleasă dimensiunea ***Year*** care admite afișarea vânzărilor numai din anii 2011-2012.

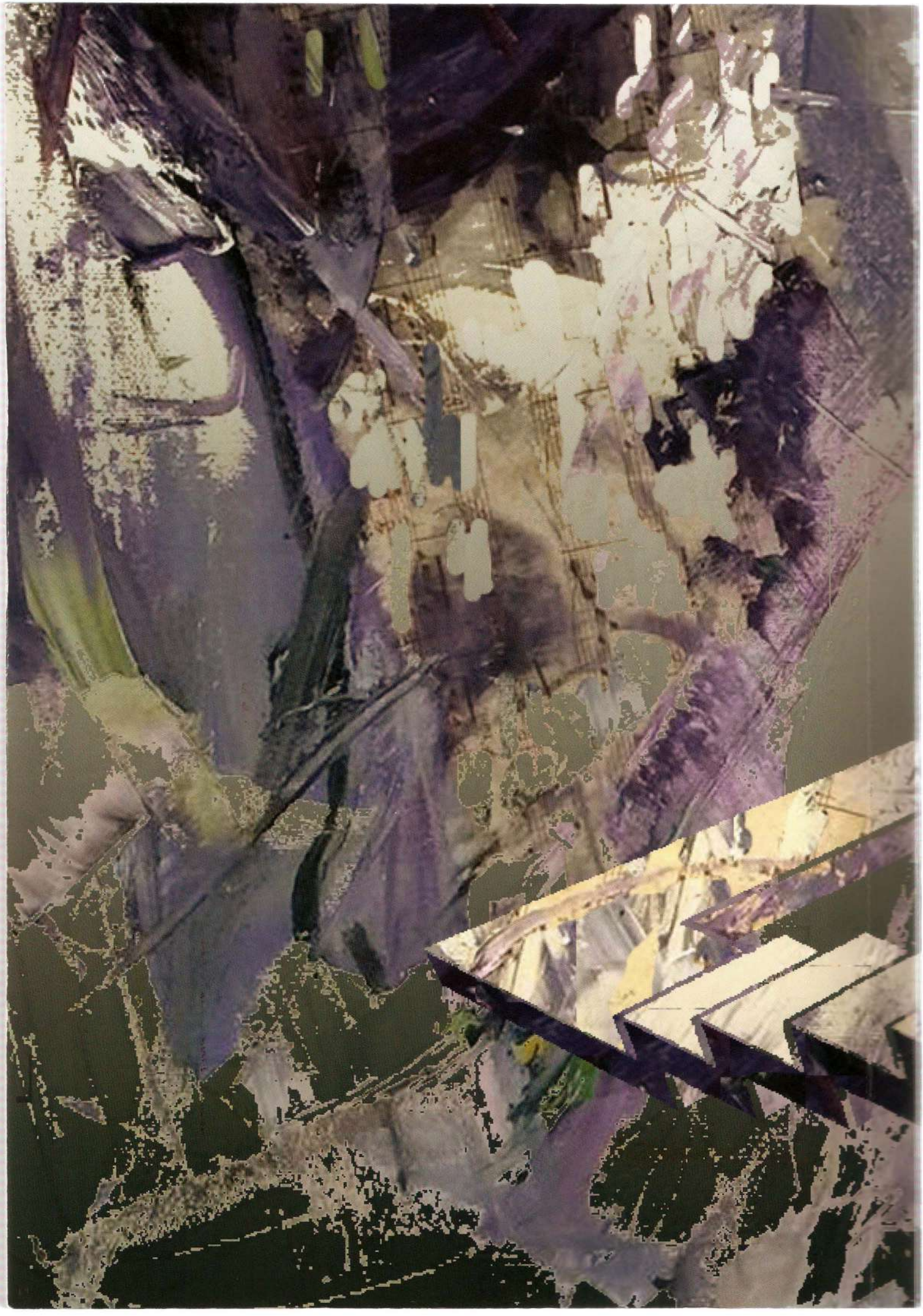
Merită atenție și câteva butoane de utilizare mai frecventă:

6. Butonul ***Auto Execute*** execută, în mod automat, interogarea și arată rezultatul de fiecare dată când se face vreo schimbare. Rezultatele sunt prezentate în panoul ***Data***.
7. Butonul ***Execute query*** execută interogarea și afișează rezultatul în panoul ***Data***.

8. Butonul ***Design Mode*** schimbă regimul navigatorului intre modul Interogare şi Proiectare.
   În cazul în care se alege regimul de interogare, în locul panelului ***Filter*** sistemul afişează codul MDX generat în mod automat.

Întrebări de control:

1. Ce este OLAP?
2. Care sunt componentele unui cub OLAP?
3. Ce sunt măsurile şi dimensiunile ce servesc drept bază de construire a acestor obiecte?
4. Care este legătura dintre un cub şi proiect al *Analysis Services*?
5. Care sunt metodele de creare a unui cub multidimensional?
6. Cum poate fi vizualizată structura cubului?
7. Care sunt particularităţile limbajului MDX, în raport cu SQL?

Sarcini practice:

1. Să se construiască un cub OLAP, utilizând, în calitate de sursă de date, tabelele ce constituie baza de date *calculatoare* şi viziunile create asupra acesteia.
2. Să se scrie câte două exemple de interogări MDX asupra cubului creat